AFRL-RY-WP-TR-2017-0163

SAFETY ON UNTRUSTED NETWORK DEVICES (SOUND)

Karen Uttecht

BAE Systems

OCTOBER 2017
Final Report

Approved for public release; distribution is unlimited.

See additional restrictions described on inside pages

STINFO COPY

AIR FORCE RESEARCH LABORATORY
SENSORS DIRECTORATE
WRIGHT-PATTERSON AIR FORCE BASE, OH 45433-7320
AIR FORCE MATERIEL COMMAND
UNITED STATES AIR FORCE

NOTICE AND SIGNATURE PAGE

Using Government drawings, specifications, or other data included in this document for any purpose other than Government procurement does not in any way obligate the U.S. Government. The fact that the Government formulated or supplied the drawings, specifications, or other data does not license the holder or any other person or corporation; or convey any rights or permission to manufacture, use, or sell any patented invention that may relate to them.

This report was cleared for public release by the Defense Advanced Research Projects Agency (DARPA) Public Affairs Office (PAO) and is available to the general public, including foreign nationals. Copies may be obtained from the Defense Technical Information Center (DTIC) (http://www.dtic.mil).

AFRL-RY-WP-TR-2017-0163 HAS BEEN REVIEWED AND IS APPROVED FOR PUBLICATION IN ACCORDANCE WITH ASSIGNED DISTRIBUTION STATEMENT.

TOD REINHART
Program Manager
Avionics Vulnerability Mitigation Branch
Spectrum Warfare Division

DAVID G. HAGSTROM, Chief
Avionics Vulnerability Mitigation Branch
Spectrum Warfare Division

JOHN CARR, Chief
Spectrum Warfare Division
Sensors Directorate

This report is published in the interest of scientific and technical information exchange, and its publication does not constitute the Government's approval or disapproval of its ideas or findings.

*Disseminated copies will show "//signature//" stamped or typed above the signature blocks.

REPORT DOCUMENTATION PAGE	*Form Approved OMB No. 0704-0188*

1. REPORT DATE *(DD-MM-YY)*	2. REPORT TYPE	3. DATES COVERED *(From - To)*
October 2017	Final	29 September 2011 – 30 June 2017

4. TITLE AND SUBTITLE
SAFETY ON UNTRUSTED NETWORK DEVICES (SOUND)

5a. CONTRACT NUMBER
FA8650-11-C-7189

5b. GRANT NUMBER

5c. PROGRAM ELEMENT NUMBER
62303E

6. AUTHOR(S)
Karen Uttecht

5d. PROJECT NUMBER
3000

5e. TASK NUMBER
YW

5f. WORK UNIT NUMBER
N/A

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
BAE Systems
600 District Avenue
Burlington, MA 01803

8. PERFORMING ORGANIZATION REPORT NUMBER

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)
Air Force Research Laboratory
Sensors Directorate
Wright-Patterson Air Force Base, OH 45433-7320
Air Force Materiel Command
United States Air Force

Defense Advanced
Research Projects
AgencyDARPA/IA
675 North Randolph
Street
Arlington, VA 22203

10. SPONSORING/MONITORING AGENCY ACRONYM(S)
AFRL/RYWA

11. SPONSORING/MONITORING AGENCY REPORT NUMBER(S)
AFRL-RY-WP-TR-2017-0163

12. DISTRIBUTION/AVAILABILITY STATEMENT
Approved for public release; distribution is unlimited.

13. SUPPLEMENTARY NOTES
PA DISTAR Case number 28440, Clearance 18 Sept. 2017. Paper contains color.

14. ABSTRACT
SOUND achieves resilient distributed systems by enabling Communities of Trust based on mutual suspicion, transparent accountability, formal methods, and differentially more reliable. The SOUND approach included: Communities of Trust: Using Introduction-Based Routing (IBR) and Reputation algorithms to dynamically establish and adapt trust levels among computational agents. This allowed well-intentioned agents to collaboratively identify and neutralize rogue agents. Accountability: Explored Accountable Virtual Machines (AVM) and developed mechanisms for supporting different levels of detailed auditing. Pillars of Trust: SOUND Communities are made dramatically more secure by having only a few trustworthy nodes on a network. Formal Methods: Proved correctness and security using formal methods to create the Simple Unified Policy Programming Language (SUPPL). The SOUND was demonstrated at PACOM, NAVSEA NSWC Research lab, and LSD-41 labs to show how it can work at scale to protect a ship network.

15. SUBJECT TERMS
Communities of trust, SAFE architecture, adaptable resiliency for computing architectures

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT:	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON (Monitor)
a. REPORT	b. ABSTRACT	c. THIS PAGE			Tod Reinhart
Unclassified	Unclassified	Unclassified	SAR	51	19b. TELEPHONE NUMBER *(Include Area Code)* N/A

Standard Form 298 (Rev. 8-98)
Prescribed by ANSI Std. Z39-18

TABLE OF CONTENTS

LIST OF FIGURES

Figure ....page

ACKNOWLEDGMENTS

The following institutions and individuals joined BAE Systems as part of our SOUND team and we would like to acknowledge their contributions to the work described herein:

Institution	*Contributors*
BAE Systems	Tim Anderson, Ray Bahr, Nicholas Brown, Robert Bryant, Tyrone Chao, David Chen, Berj Chilingirian, Silviu Chiricescu, Greg Eakman, Russell Elliott, Joseph Fahey, Kendall Farnham, Michael Figueroa, Victor Firoiu, Karl Fischer, Chris Frederickson, Martha Gonsalves, Hillary Holloway, Suraj Iyer, Andrew Kaluzniacki, Basil Krikeles, David Lebling, Clifton Lin, Andrew Macbeth, Josh McGrath, Ed Pelletier, Balasubramaniam Ravichandran, Jothy Rosenberg, Howard Rubenstein, Andrea Ruggiero, Kathy Ryall, Deborah Stott, Greg Sullivan, Andrew Sutherland, Yelena Taytslina, Michelle Torrelli, Peter Trei, Karen Uttecht, Christopher White
University of Pennsylvania	Arthur Azevedo de Amorim, Andre DeHon, Andreas Haeberlen, Ben Karel, Eric Keller, Albert Kwon, Alex Marder, Jonathan Smith, Antal Spector-Zabusky, Steve Zdancewic
Portland State University	Robert Dockins, Andrew Tolmach
Apogee Research	Greg Frazier, Anders Jagd
Good Harbor Security Risk Management	Richard Clarke, Lisa Davis, Jacob Gilden, Emilian Papadopoulos, Evan Sills
Red Balloon Security	Calvin Chu, Ang Cui, Peter Minzicu, Joseph Pantoga
Consultants	Jason Bloomberg, Patrick Harding, David Linthicum, Thomas Knight, Greg Morrisett, Andrew Myers, Benjamin Pierce, Olin Shivers

1.0 SUMMARY

The Department of Defense (DoD) faces threats from both outside and within, and no single system can be reliably secured against all threats. This is further complicated by the fact that missions operate over large, heterogeneous sets of computing resources, potentially from multiple jurisdictions. Insider threats are increasingly prevalent, and one cannot assume that because something or someone is allowed within the perimeter, that they may not be the source of a future attack. Additionally, operators are inundated with alerts from many different sensors, making processing the massive amount of information in a timely manner difficult.

The Safety On Untrusted Network Devices (SOUND) system provides a secure, resilient, scalable distributed computing platform upon which mission-oriented distributed applications can be built that are both resistant to and capable of adapting to an attack to continue a given mission.

The SOUND program investigated bringing together three promising technologies: Accountable Virtual Machines (AVM), Quantifiable Trust Management (QuanTM) and Introduction-Based Routing (IBR). Over the course of the program, we found that QuanTM was far too complicated for the needs of the military, and that AVM was not able to protect against exploits which utilized application flaws.

As a result of these discoveries, we chose to shift directions a bit and combine IBR with a set of SOUND Services: SOUND Trust Service (STS), SOUND Policy Service (SPS), Sound Identity Service (SIS), SOUND Reputation Service (SRS), SOUND Data Service (SDS), and a SOUND Connection Service (SCS), based on IBR. In addition to these services, we developed a SOUND Sensor Framework (SSF) to integrate sensors and SOUND Administration Console (SAC) to allow for administration of the system.

We did research on existing reputation algorithms and chose to base the SOUND Reputation Service on the REGRET algorithm. By integrating the reputation service with IBR, we were able to create secure Communities of Trust (CoTs), inside which users and hosts would have a reputation which tracked them throughout the network. We were able to provide accountability and monitoring through our Sensor Framework and by utilizing reputation to isolate suspicious members from the CoT. Members of the CoT could have different levels of trust for other members, providing a differentially reliable decentralized trusted environment. SOUND is able to incorporate existing commercial or government sensors alerts into the reputation. We found that reputation provided a way for operators to view the health of the network in a simple way which reduced the alarm fatigue.

We experimented with how we could build Pillars of Trust – nodes with a higher level of protection by performing experiments through simulation showing how if a few nodes in the network had a higher level of trust, it enhanced SOUND's ability to contain the attack. We built a prototype of a SOUND Router which is built on a formally verified secure processor.

In order to define policies within SOUND, we created a SOUND Policy Service and the Simple Unified Policy Programming Language (SUPPL), which allows formal specifications of policies in a way that allows conflicts between policies to be discovered before they create problems in deployment.

We demonstrated the SOUND System to the Pacific Command (PACOM), Defense Information Systems Agency (DISA), Special Operations Command (SOCOM), as well as deploying it in several of the shipboard control system labs at NAVSEA Naval Surface Warfare Center (NSWC) in Philadelphia. We integrated the SOUND System with another MRC project, Network Path Diversity (NPD) and showed how the two systems were able to secure Navy shipboard networks. NAVSEA agreed to sign a Technology Transfer Agreement (TTA) to deploy SOUND to a ship.

Our recommendation is that the government continue pursuing deployment of SOUND to a Navy vessel for testing and to the fleet. We further recommend that SOUND be utilized for securing any Information Technology (IT) or Industrial Control System (ICS) network from intrusion, and would still be useful for SOCOM or DISA to consider.

2.0 INTRODUCTION

2.1 Problem Description

Computational support for today's DoD missions operate over a large, heterogeneous, distributed set of computing resources—from personal mobile devices to massively parallel computers managing millions of connections and petabytes of data. These distributed components must cooperate across agencies and across coalitions of allies; each partner brings independently-managed systems of varying reliability and trust into the distributed resource mix, and each has different policies and legal restrictions.

Today, we cannot reliably secure any single system against cyber attacks, even when it is wholly owned by a single agency with a single mission. Computations can be disrupted (denial-of-service); machines can be co-opted (taken over and used by attacker); data can be corrupted and stolen. The problem is even further beyond the state-of-the-art when considering a coalition of machines under different jurisdictions. There is currently no principled way to describe what such systems should be doing and thereby differentiate proper and compliant agents from rogue actors.

In addition to threats from outside of an organization, we are also faced with insider threats. It is not enough to simply protect the boundaries of our networks, we must assume that the threat can come from within.

Ineffective legacy practices have failed to counter contemporary information security and privacy threats. Modern IT operates on large, heterogeneous, distributed sets of computing resources, from small mobile devices to large cloud environments that manage millions of connections and petabytes of data. Protection must often span organizations with varying reliability, trust, policies, and legal restrictions. Centrally managed, host-oriented trust systems are not flexible enough to meet the challenge. New research in distributed and adaptive trust frameworks shows promise to better meet modern needs, but lab constraints make realistic implementations impractical.

Safety On Untrusted Network Devices (SOUND) provides secure, resilient, scalable distributed computing platform upon which mission-oriented distributed applications can be built that are both resistant to and capable of adapting to an attack to continue a given mission.

2.2 SOUND Solution

Safety on Untrusted Network Devices (SOUND) is a new security platform built from the ground-up to verifiably protect communications up to and within applications. Legacy solutions are compartmentalized, focusing on just the systems and networks that carry traffic, independent of content, or are application driven, disregarding behavior that occurs at lower levels of the network stack. SOUND unifies the tools organizations already have in order to proactively protect communications, separate outsiders from insiders, and track misbehavior. SOUND's primary assumption is that the technical environment is vulnerable to attack. It trusts applications, their users, and the subsystems applications and users operate on, only as long as they behave appropriately [FUJ15].

The SOUND research goal is to create a digital immune system that can remember the damage caused by communications behavior and automatically reacts to contain that damage and its source. SOUND eliminates digital anonymity by establishing a “Community of Trust” (CoT) in which identifiable members are monitored for their adherence to defined rules of behavior. Each CoT embodies three common resilience characteristics, including:

- *Accountability* – Compare individual behavior against a common community policy. This capability was initially based on the Accountable Virtual Machines (AVM) technology [6].
- *Protected Communications* – Anchor network communications and allow members to report on community interactions. This capability evolved from the Introduction Based Routing (IBR) technology [7].
- *Reputation* – Provide a framework for measuring member reputation. This capability was initially based on the Quantitative Trust Management (QuanTM) technology [8].

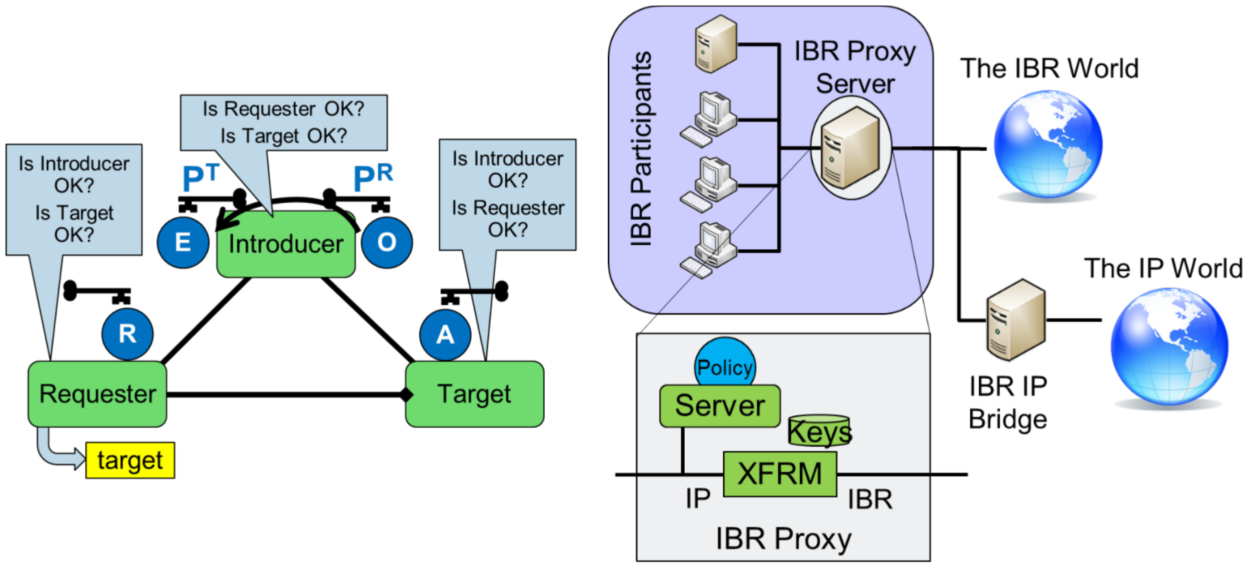
2.3 Background

2.3.1 Introduction Based Routing

Given enough attempts, even a low-probability cyber attack (e.g., spam, password guessing, Structured Query Language (SQL) injection, port scanning, Distributed Denial of Service (DDoS)) will eventually succeed. As attackers rattle the proverbial doorknob looking for a way in, each failure is observable and should serve to identify them as attackers, but, in today’s networks, they usually go unidentified, and there are no repercussions for such misbehavior.

In contrast to the *bank vault model*—preventing malfeasance via (metaphorical) armor plating which strictly controls what an actor can and cannot do, ***Introduction-Based Routing*** (IBR) follows the *shopping model*, in which each individual actor identifies safe partners with whom to conduct business. When we walk into a store to make a purchase, it is with some confidence that we will not be “ripped off”, because the store does not want to risk losing all the future business of not only that customer, but the customer’s friends, as well. Choosing where to shop is a *repeated game*; misbehavior in one round impacts the reputation of involved parties in subsequent rounds, and their subsequent willingness to interact. Unfortunately, within current Internet Protocol (IP) standards, every interaction is a *single-play* game. Furthermore, we do not currently use information from other, trusted nodes to assist with establishing reputations.

IBR is an active network defense that changes the economics of cyber attacks by requiring that new connections be formed by way of “Introductions.” Figure 1 (a) shows the introduction handshake: 1. Requestor sends (R)equest to Introducer. 2. Introducer sends (O)ffer to the Target. 3. Target sends (A)cceptance to Introducer. 4. Introducer sends (E)stablished to Requester. The connection is established and exists until closed by either party. Figure 1 (b) shows a notional architecture of how IBR is deployed on a network. The IBR Proxy Server allows hosts to participate in IBR simply by routing their packets via the proxy server. The IBR IP Bridge connects the IBR network to the Internet. It acts as an Introducer to the Internet nodes, causing them to have a persistent reputation in the IBR network.



(a) The Introduction Handshake (b) Notional IBR Architecture

Figure 1: Introduction-Based Routing Active Network Defense Solution

By collecting positive and negative feedback about interactions, IBR participants (independently) assesses the trustworthiness of other participants. Nodes can decide for themselves whether to participate in connections, whether as introducers or endpoints. Misbehavior not only degrades the reputation of the bad actor, but also of those hosts that were willing to introduce it. When a system's reputation is poor, it will be *isolated from the network*. IBR is a locally-managed, peer-to-peer federation of nodes where observations and reputations are not shared (other approaches, like eBay's, rely on centrally-managed reputations).

The basic idea of establishing a connection via introductions is illustrated in Figure 2. When Host A wishes to establish communications with Host E, it contacts a few "neighboring" hosts that are well trusted based on past experience. These hosts (such as Host B) provide introductions to other hosts (such as Host C) until A is put in touch with Host E. This remembered pedigree allows Host A to hold both the end host and the introducers accountable for poor behavior. At any given point in this process, introducing hosts may refuse to offer an introduction for hosts that they do not trust, thus isolating bad actors from the system. To get the system started, each node must have some a priori connections, established outside of IBR.

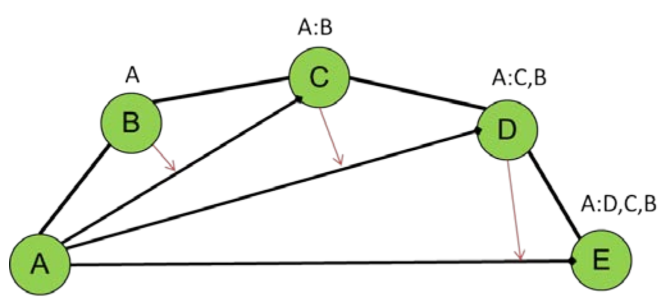


Figure 2: Introduction of Host A to Host E via B, C and D

Although IBR provides a modest set of misbehavior sensors, IBR emphasizes integration with each site's existing set of commercial misbehavior sensors.

The IBR Protocol does not specify the policies by which participants label behavior as malicious and update their own opinions of other participants, and can operate with large variations in such policies across the network. Four classes of effective policies discovered over the course of this research is based on (a) playing tit-for-tat—reporting as a bad actor anyone who reports you as a bad actor; (b) incrementing and decrementing one's reputation of an interaction partner for good and bad behaviors at different rates, which must be roughly tuned to detect false alarm rates; (c) adjusting the reputation of the subject of a report more when the reporter has a high reputation; and (d) forgiving bad behavior—allowing a bad reputation to recover—over time.

IBR is implemented as an Internet Protocol (IP) based on a small number of message types, corresponding to such events as offering or accepting an introduction, closing a connection, or sending post-connection feedback to an introducer. IBR uses the IPsec tunneling protocol, and most messages use public/private key encryption to verify endpoint identity.

By design, IBR is a "drop in" solution, with low barriers to adoption. It can be adopted and deployed piecemeal, is open sourced, scalable, has low latency cost, requires no central administration, and require no changes to the entrenched infrastructure.

Simulations and game theoretic analyses over a wide range of network topologies, numbers of attackers, detector false positive and negative rates, and policy parameters provides evidence that widespread adoption of restrictive introduction and acceptance policies is strategically stable; that IBR adoption need not be universal to sharply reduce misbehavior, including "reputation attacks"; and that IBR is difficult to "game".

2.3.2 Quantitative Trust Management (QuanTM)

A major challenge in determining trust in a decentralized setting is selecting robust Trust Evidence. How do we know who is making a request and if they are authorized to perform such a request? With what confidence can we draw these conclusions? Such evidence can be static, such as possession of cryptographic keys or a cryptographically signed certificate as in the original Trust Management proposal (implemented in KeyNote [BFK98]). However, as we deal with varying policies among different agencies assembled to contribute to a common mission, the evidence must be context-dependent, as in Dynamic Trust Management (DynTM) [BKL+09], with its sophisticated, context-sensitive authorization policies. When agents are subject to compromise and when we must continually make judgements about new agents in a system authorized by decentralized authorities, the evidence and judgements may need to be policy-, context- and behavior-dependent, as in Quantitative Trust Management (QuanTM) [WAC+09].

Figure 3 represents the QuanTM architecture. The three boxes (demarcated with dashed lines) represent the Trust Management (TM), Reputation Management (RM) and Decision Management (DM) subsystems of QuanTM, moving from left to right in the figure.

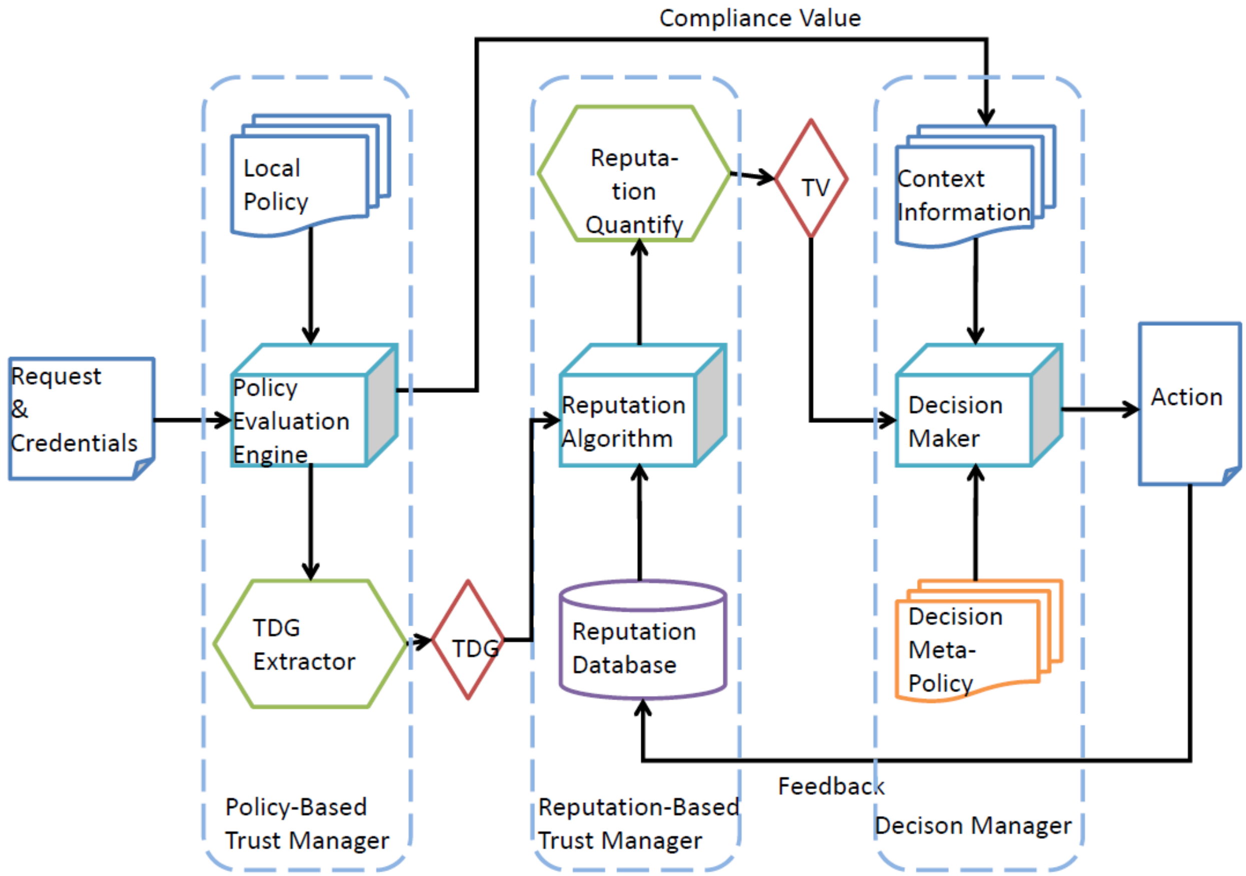


Figure 3: Architecture for Quantitative Trust Management

On the left side, a request and credentials authorizing that request are presented. An example credential is represented in Figure 4.

```
Authorizer: SecDef
Licensees: SecARMY || SecNAVY || SecAIR
Conditions:
operation == "query" -> "True";
operation == "update" -> "Maybe";
Signature: "rsa-sig:1294..."
```

Figure 4: An Example Keynote Credential

This credential represents that an authorizer SecDef authorizes two operations query and update, to three Licensees, for some database access. Compliance values are computed using the conditions values, with True meaning trust, False meaning don't trust, and Maybe meaning the system must consult additional policy rules to make a trust decision.

The TM subsystem parses the KeyNote language and computes a Compliance Value (CV) that is passed (over the RM subsystem in the figure) to the DM subsystem. The TM subsystem also constructs (using the dependencies inherent in delegated authorities) a trust dependency graph

(TDG) annotated with compliance values computed using local policies. This TDG is what is passed to the RM subsystem.

The RM subsystem consults observations stored in a Reputation Database that represent learned knowledge in QuanTM. A Reputation Algorithm is applied to the TDG object, computing trust values (TVs) using the KeyNote Compliance Values and the behavioral data from the Reputation Database; the Reputation Quantifier produces a final Trust Value (TV), which is passed to the DM subsystem. In the QuanTM paper, the Reputation Database was assumed to be pre-populated; in our proposed SOUND effort, an initial database and feedback mechanism derived from IBR and audit will flow into the system at the arc on the lower right, feeding into the Reputation Database.

In the DM subsystem, the CV is combined with the context information (e.g., indications of a high alert) before being passed to the Decision Maker. This injection of context is where the Dynamic Trust Management is achieved; the Decision Maker combines the information from the TM subsystem and the Reputation Management subsystem using a Decision Meta-Policy and determines an Action, shown emerging from the right hand side of the figure. The Decision Meta-Policy is application-dependent, but might represent the application's preferences for trust evidence and how these are affected by context and observed behaviors. Feedback on the action is incorporated into the Reputation Database.

Chang, et al. [CVW+11] have demonstrated the application of a behavior-based reputation system to the Border Gateway Protocol (BGP), a highly decentralized information exchange for Internet path information. The derivation of probabilistic measures of trust for Autonomous Systems (ASs) on the Internet is shown to be useful to increase routing performance and dampen the effect of misbehaving nodes.

2.3.3 AVM

Accountable Virtual Machines (AVM) provides the capability to audit software system executions through logging the execution and comparing it to a known-good version. This auditing system does not require trust in the hardware or the accountable virtual machine monitor on which the binary executes. AVMs provide users with the capability to detect faults, identify faulty nodes and to provide evidence of which machine caused the fault. The AVM must maintain a log with enough information to reproduce the entire execution and cryptographically record each outgoing message to link it to the execution log. AVM then detects faults by replaying the execution using a known-good copy of the binary and checking the visible behavior is identical to the previously run version. AVMs can do this with any binary image that can be run inside a VM.

If we are going to build a Community of Trust out of a population that includes malicious and compromised agents, how do we establish trust? If agents trustworthiness can change over time (e.g. they are compromised, or a sleeper agent awakens), how do we detect misbehavior and reassess trust?

In order to check whether a system is performing satisfactorily, each element of the system must provide tamper-proof evidence. Computational elements range from individual instructions on a

host, to functions, to processes, to ensembles of processes on a single host, to a host as a black box, to distributed ensembles. [HKD07, HARD09, HK09, BDHU09, HARD10, Hae09].

In an accountable system, each node is responsible for performing some specific function, such as storing files or forwarding packets. This function is called the node's expected behavior. However, it is assumed that some nodes may deviate from their expected behavior, e.g., because they are faulty or have been compromised. The goal of accountability is a) to detect when such misbehavior occurs; b) to identify misbehaving nodes; and c) to produce evidence that irrefutably links the misbehavior to a specific node.

Existing accountability techniques [HKD07, HARD10] generate evidence as follows. First, each node is required to maintain a tamper-evident log of its local actions, such as sending or receiving messages, certain processing steps, etc. The logs of different nodes are intertwined such that, if a node tampers with its local log or maintains an incomplete log, at least one other node is guaranteed to detect this. Second, each node is associated with a set of witnesses (auditors) that periodically audit that node's log. Thus, as long as each node has at least one correct witness, misbehaving nodes cannot escape detection: If they misbehave but maintain a correct log, the witnesses will discover the misbehavior; if they attempt to cover their traces, this will be discovered.

Figure 5 illustrates this approach in the context of a multiplayer game. Each of the three players maintains a tamper-evident log of his or her actions; Eve has modified her game software to gain unlimited ammunition (left). When Alice becomes suspicious of Eve's good performance, she can audit her log and check it for tampering (middle); she then replays the log using her own copy of the game software to check for cheating (right). Since Alice's copy of the game software has not been modified, the behavior during replay will be different. Alice now knows that Eve must have been cheating, and she can use the copy of Eve's log to convince Bob.

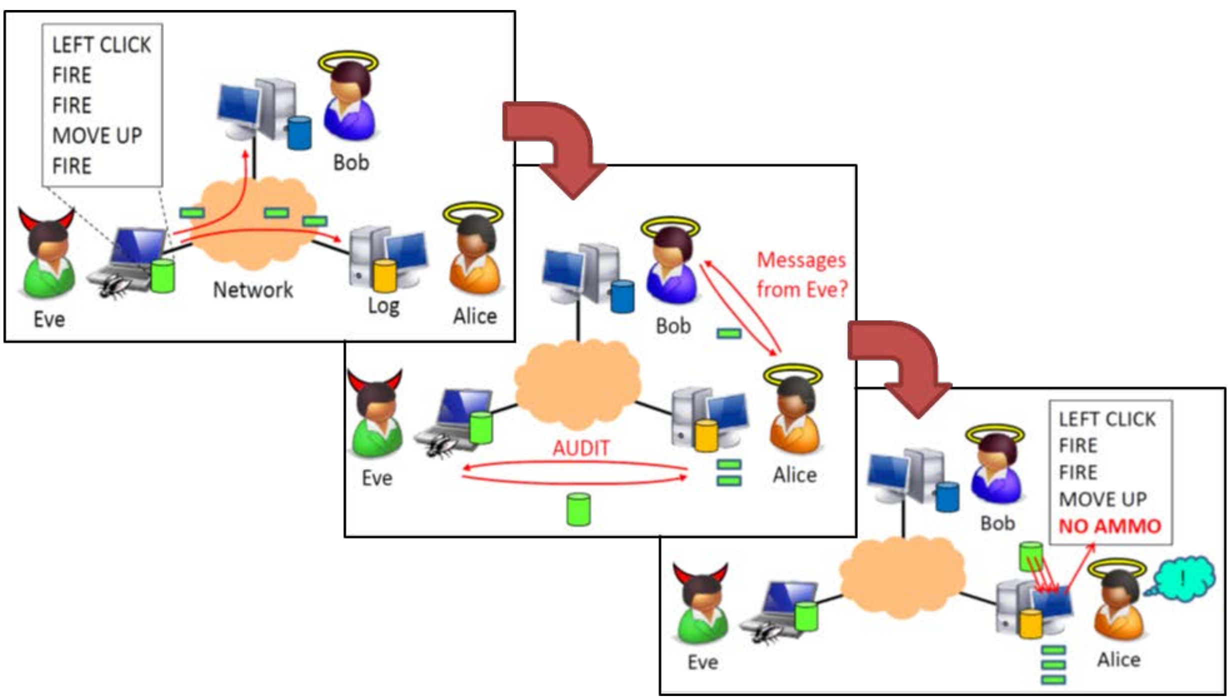


Figure 5: Simplified Example: Accountability in a Multiplayer Game

3.0 METHODS, ASSUMPTIONS, AND PROCEDURES

SOUND (Safety On Untrusted Network Devices)'s goal is to provide a secure, resilient, scalable distributed computing platform upon which mission-oriented distributed applications can be built that are both resistant to and capable of adapting to an attack to continue a given mission.

In order to achieve this goal, we provide two mechanisms:

1. *Communities of Trust* that provide an "immune system" for detecting and adapting to misbehaving elements of the distributed system, and
2. Highly reliable and trustable infrastructure nodes from which we can bootstrap a reliable "public health infrastructure" and provide oversight for the large number of mutually suspicious nodes in the network.

3.1 Research Methodology

Our research and development methodology followed an Agile/Scrum approach where design and research direction were motivated by a number of "User Stories", which described how our eventual users would want to use the system, and how attackers would try to attack it. We built the system up by focusing on one type of attack by a persistent malicious adversary per sprint. As we got further along in the project, our approach was heavily motivated by what we learned from our transition partners. We would demonstrate systems to potential transition partners, get feedback on what was useful to them and what was not, and use that to determine what the next steps should be. User stories have been documented in Appendix F of the SSDD.

As a result of this methodology, our methods for researching solutions to the problem evolved over time. In section 3.2, we describe the initial approach and in section 3.3, we describe what the approach evolved to become.

3.2 Baseline System: Integration of IBR, QuanTM, and AVM

Our baseline system merged three promising technologies: Introduction Based Routing (IBR), Quantitative Trust Management (QTM), and Accountable Virtual Machines (AVM).

For detecting and adapting to misbehaving elements in a heterogeneous distributed system, our dynamic *Communities of Trust* was based on a novel integration of research in *Accountability, Reputation, and Protected Communications*. Initially based on three founding technologies – *Accountable Virtual Machines (AVM), Quantitative Trust Management (QTM), and Introduction-Based Routing (IBR)* – our research efforts extended beyond those technologies to develop a more powerful and practical SOUND implementation that would support real operational environments.

In more detail, SOUND a breakthrough in security and resiliency of heterogeneous networked systems by innovating in the following four areas:

1. **Communities of Trust**: SOUND implements an innovative combination of Introduction Based Routing with Quantitative Trust Management. This combination was intended to support a self-organizing, dynamic web of trust, backed by context-sensitive policies. This

would solidify distributed trust management by extending formal methods developed under the SAFE program to the SOUND communications layer, resulting in dramatically more trusted systems than is possible currently. The combination of these technologies would allow SOUND to implement unforgeable communications, such as connections between distributed processes.

2. **Accountability and Monitoring with Mutual Suspicion**: The plan was to have SOUND apply a fine-grained protection model that focuses on the *root cause* of misbehavior instead of just the system that is hosting the damage. For this to work, SOUND would implement an innovative *multi-layer attribution* technique that will allow SOUND to attach sensor observations to all of the identities involved in a detected misbehavior event. We also investigated extending work by Haeberlen [HARD10, Hae09] to enable tamper-evident logs and audit trails on endpoint hosts, acting as a SOUND sensor to enhance accountability with the host, however we found that this didn't provide protection for application flaws.
3. **Pillars of the Community: Establishing a differentially reliable decentralized trusted environment:** The SOUND project took on a two-pronged approach to enhance the inherent trust in an operational environment. To push SOUND resilience properties closer to the endpoint hosts without causing significant impact to host functions, SOUND needed to implement a distributed trust management capability that monitors user and host activities. This will allow a more rapid deployment of reputation management in any communications environment. Then, SOUND development would extend the SAFE implementation from the CRASH program to allow SAFE hosts to operate in a heterogeneous distributed system of differentially reliable elements (like accredited public health system hospitals). SAFE-based elements would consist both of general purpose (but highly trusted) compute servers running critical computing tasks (e.g. auditing, introducing), and also of embedded devices (e.g. routers) obtained by creating specializations of the SAFE architecture. A distributed set of SAFE-based nodes will provide a Trusted Computing Base (TCB) for any MRC system.
4. **Formal Specification and Model-based Programming**: SOUND investigated implementing formal specifications for all services, protocols, and computational platforms in a SOUND system that provide interfaces for external subscription to SOUND data and to provide feedback into SOUND decisions. To support programming of policy and sensor control, SOUND would also include a high-level policy language that allowed direct specification of desired system states and reduces conflict between SOUND control authorities.

3.3 New Focus: Reputation

After achieving minimal results in attempts to merge the SOUND technologies into a comprehensive framework, the SOUND team modified its approach to emphasize development of a research platform that could compute reliably in the face of attacks known and unknown on a differentially trustable set of distributed compute nodes with variable susceptibility to attack. This new approach sought to apply an agile development methodology for building a new SOUND platform that would implement and extend the key research technologies in alignment with a series of plausible attack scenarios with dangerous potential consequences to explore the platform's resilience response. This new SOUND platform is based on implementing an

Observe-Orient-Decide-Act paradigm through a set of networking, authentication and trust services that define and enforce Communities of Trust, providing an immune system against attacks.

A shift towards transition caused us to further modify our approach to emphasize SOUND features that will support demonstration to PACOM and other capabilities that build a broader transition story. SOUND reputation will continue to evolve, but we now expect development to be limited to a single SOUND community with constrained reputation portability. We shifted emphasis to address the security considerations of the SOUND platform itself and focus on building stronger core capabilities.

Our modified approach innovated in the four areas in the following ways:

1. **Communities of Trust**: Deeper investigation into utilizing QuanTM for trust management in SOUND found that the architecture was much too complex and couldn't be adapted easily to tracking variable trust, instead of determining complex static trust. The IBR reputation system was designed with network security in mind, but didn't have a concept of a Community of Trust, each IBR node operated independently. Our discussions with transition partners made it clear that centralized control and communities would be essential to success, but so would a system which was understandable enough to be useful to operators. As a result we chose to extend a reputation algorithm, Reputation in Gregarious Societies (REGRET) [SS01], originally designed for reputations in shopping and review systems, to our needs. The REGRET algorithm combines four components: node A's opinion of node B, node A's opinion of node B's community, node A's community's opinion of node B, and node A's community's opinion of node B's community. In addition, the method of incorporating sensor data allows for different classes of behavior to affect the reputation and decay using definable functions.
2. **Accountability and Monitoring with Mutual Suspicion**: In choosing to base the SOUND reputation scheme on the REGRET algorithm, one of our goals was to define a reputation system which would allow *root cause* to be determined. We also investigated how SOUND identities could be attributed to hosts and applications, in addition to individuals. Our approach was to create a wrapper which would do the appropriate authenticate on behalf of the application and individual in order to allow legacy applications to be used.
3. **Pillars of the Community: Establishing a differentially reliable decentralized trusted environment:** We experimented with how adding provably trustworthy nodes to the network could bolster the effectiveness of SOUND in a future integration.
4. **Formal Specification and Model-based Programming**: As our research took shape, we found that one of the more difficult research problems we needed to address was how to detect conflicts within a set of policies. Our high-level policy language needed to account for sets of policies written by different individuals at different times and allow us to formally reason about policies.

3.4 Assumptions

In our work on SOUND, we performed our research under a set of assumptions about the environment SOUND would be operating in, outlined below:

- **Insider Threat:** We assumed that attacks could come from within the network, from identities and individuals which were considered trusted.
- **Differential Trust:** Entities communicating using SOUND may have differential levels of trust in one another. We needed to build a system that would allow less trustworthy agencies to connect.
- **Zero Day Attacks:** We assumed that the attacks may not be ones that could be identified by signature, but that would be visible by a pattern of multiple behaviors over time.
- **Centralized Control:** In designing SOUND, we assumed that the organization deploying it would want control over the entire Community of Trust, as we discovered in discussions with military agencies.
- **Legacy Applications:** Originally we assumed that applications would be programmed or retrofitted explicitly to work with SOUND. We modified this assumption after discussions with transition partners revealed that this would be extremely difficult to accomplish. Our later work assumed that existing applications and devices would not be modified.
- **Automatic vs Human-in-the-loop:** Another original assumption was that SOUND would operate entirely autonomously, making decisions based on policy and taking action accordingly. Discussions with transition partners made it clear that until our technology had been proven and in regular use, transition partners would need a human in the loop, choosing whether to act on SOUND's recommendations.
- **Authentication:** SOUND relies on authentication of users, but our research was not focused on authentication systems, so we assumed that a deployed system would be integrated with the customer's authentication.
- **Encryption:** SOUND utilized IBR's integrated IPSec tunnels for encryption, but we also assumed that a transition partner would specify which encryption technologies were needed and integrate them into SOUND.
- **Unicast vs Multicast:** We assumed through most of the project that communication over the SOUND system would be done via unicast protocols (such as TCP, etc.). When we initially worked with NAVSEA NSWC, we found out that most communications on shipboard systems are done via multi-cast.

3.5 Red Team

As part of our testing, we planned to conduct Red Team assessments. In these assessments, a group of engineers from BAE who worked in the Cyber & Communication Technologies Group, but not on the SOUND project, would review the code, design and perform attacks against a live system to internally assess its vulnerabilities.

4.0 RESULTS AND DISCUSSION

We have organized the results into four main sections. First, a discussion of experiments performed. Second, a discussion of the SOUND Platform. Third a discussion of the tests run. Lastly, the demonstrations and lab deployments done during the program.

4.1 Experiments

4.1.1 SAFE+SOUND Experiment

SAFE, developed under the DARPA CRASH program and SOUND were designed to be companion projects. SAFE focused on the processor and the host, SOUND focused on the network. We wanted to see how much having a SOUND node running on a SAFE processor would make a difference in our ability to thwart attacks and whether the two systems would provide benefits to one another.

SAFE protects only the machine its running on, not the network. It also can’t protect from insider attack, or bugs within an application. It does protect against the top 20 CWEs (such as buffer overflow).

In order to do this, we ran simulations to see if SAFE bolstered SOUND and we found that a few SAFE machines in the network were able to increase the overall security of the network significantly.

4.1.1.1 Experiment Design

For this experiment we utilized an IBR simulator, which simulated the behavior of IBR and attacks to see how they would spread within a network. The network was divided into a series of enclaves, each enclave had a simulated SAFE machine within it. The enclaves were connected via a backbone of routers. A single node would be deemed compromised and would then attempt to choose another machine to attack and attempt to attack it. A non-SAFE machine could be compromised, but would have a percent chance of detecting the attack. A SAFE machine was considered unable to be compromised, but the compromised machine would still attempt to attack it and the SAFE machine would therefore report the attack, affecting the reputation of the attacker. Figure 6 shows the setup for the experiment and Figure 7 describes the scenario, variables and assumptions.

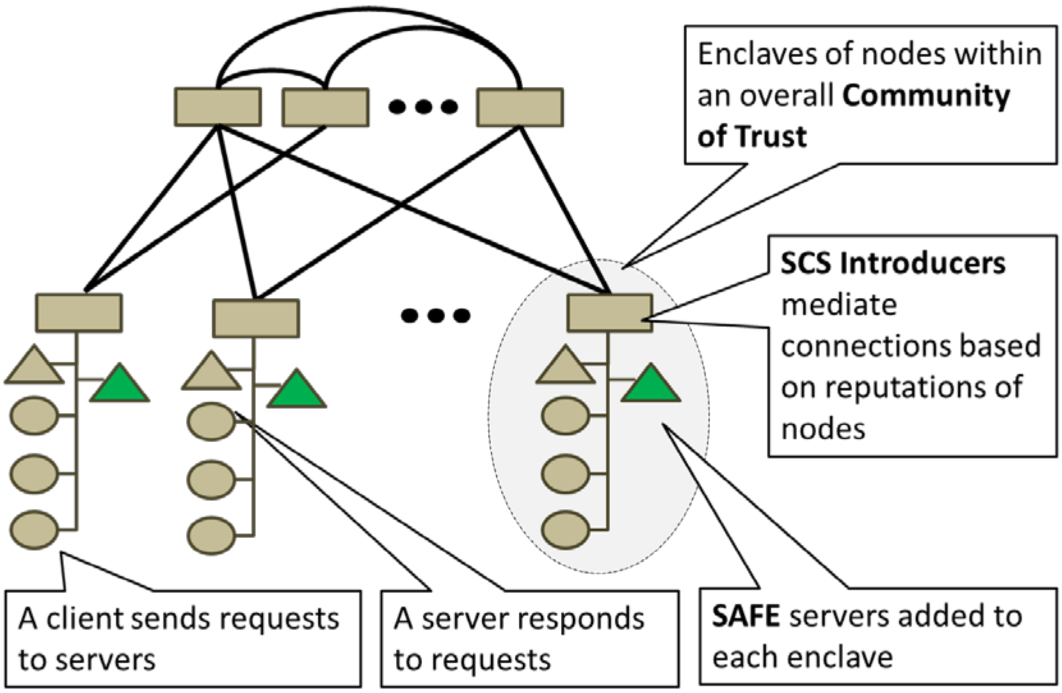


Figure 6: SAFE and SOUND Experiment Setup

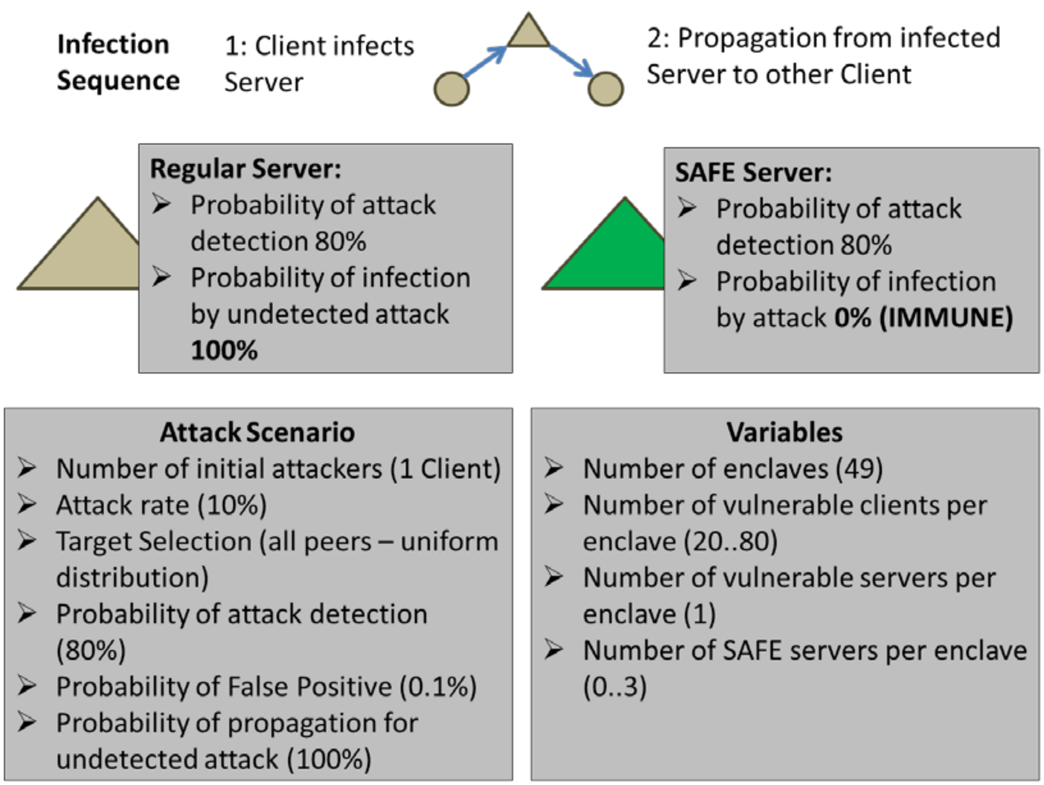


Figure 7: Experimental Setup, Scenario, Variables, and Assumptions

4.1.1.2 Results

What our experiment found was that a small number of SAFE machines made a great difference in how quickly the IBR system was able to contain the attack. (See Figure 8) The SAFE nodes acted as observers, quickly noticing and reporting the behavior of the infected nodes, which allowed IBR to stop accepting connections from them. This kept the infection contained to a few enclaves, instead of spreading across the network.

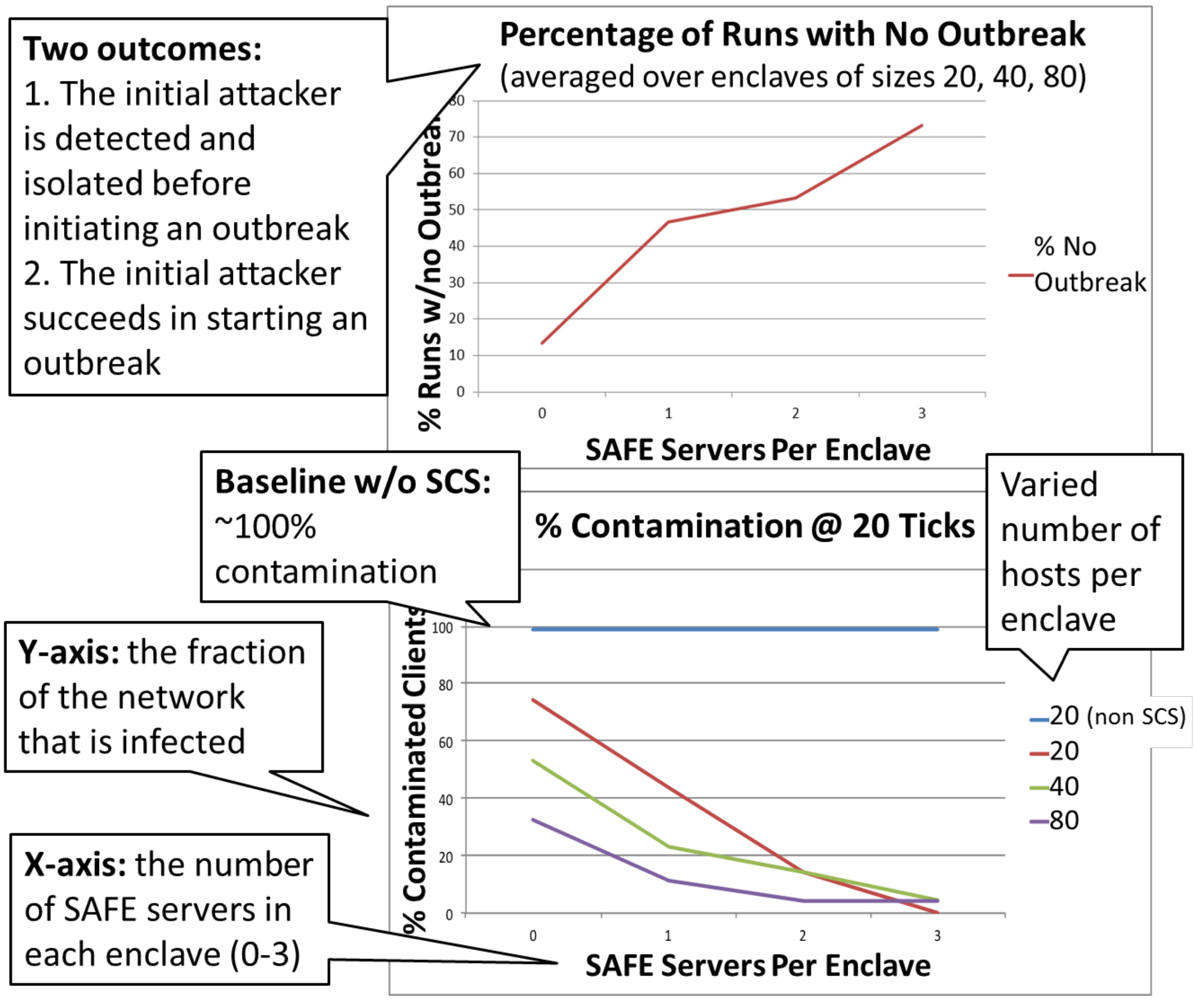


Figure 8: Rate of Contamination With and Without Added SAFE Servers

However, this experiment assumed the attacker did not know which nodes were SAFE nodes. It would make sense that if the Adversary had that information, they would program the malware to avoid the SAFE nodes, therefore reducing the ability for SAFE to protect the network.

Additionally, we realized that this benefit is also true of heterogeneity of the network. In a network where all nodes are running identity operating systems, malware is capable of attacking and compromising all of them. But if nodes are heterogeneous, presumably the malware isn't able to compromise all of them, but rather a subset. The nodes which are running operating systems not susceptible would act similar to the SAFE nodes in our experiment.

4.2 SOUND Platform

This section discusses the resulting SOUND platform.

4.2.1 SOUND Services

SOUND is comprised of a set of distinct and independent services that enhance trustworthiness by maintaining logical "Communities of Trust" (illustrated in Figure 9) and tracking the reputation of all community members. Those services reduce reliance on perimeter defenses for protecting mission operations and enhance control over how information flows across community boundaries. Their functions are founded on the premises that distributed, cloud-oriented applications will always have flaws and that SOUND cannot completely trust insiders or endpoints. Rather, SOUND will operate on a concept of differential trust where each community member is judged based on its individual actions. The architecture increases attacker workload through necessity of being a recognized and legitimate community member and of mounting a coordinated attack against an application and the SOUND platform simultaneously. We designed the SOUND platform to be a dynamic immune system of services that are capable of controlling the communications channels between all community members. Those services collectively support the SOUND Observe, Orient, Decide, Act (OODA) loop, providing the mechanisms needed to collect sensor information, to interpret sensor observations to detect potentially damaging behavior, to process those observations against policy to decide how to respond to the behavior, and to protect the community from the behavior while maintaining mission effectiveness.

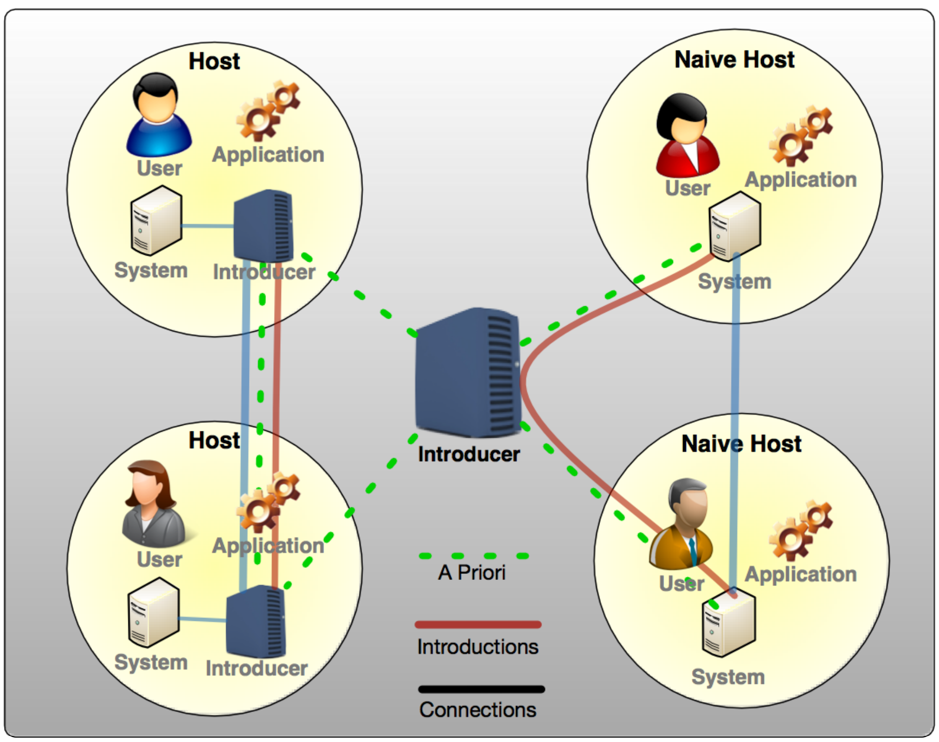


Figure 9: Community of Trust

SOUND is comprised of the following components / services:

- **SOUND Trust Service (STS)**: STS is the central SOUND decision-making and policy enforcement engine. STS receives sensor observations and acts on those observations by applying predefined policies to manage connections, identities, sensor behavior, and notification of community participants. See Appendix B of the SSDD for more detail.
- **SOUND Policy Service (SPS)**: SPS stores and manages SOUND policies. Policies managed by SPS include those defining communities, interactions between SOUND communities, and reputation distribution across SOUND nodes. See Appendix C of the SSDD for more detail.
- **SOUND Identity Service (SIS)**: The common identity management service for all community participants, providing credentialing and authentication. See Appendix D of the SSDD for more detail.
- **SOUND Reputation Service (SRS)**: SRS captures and stores Observations as submitted by community participants, and uses those Observations to calculate whether or not one entity trusts another. See Appendix K of the SSDD for more detail.
- **SOUND Data Service (SDS)**: The SDS is intended to be a middleman between the application and the data provider, such as a SQL database, which enforces and maintains permissions based on SOUND policies and reputations. As this piece was of less interest to our transition partners, it was later dropped from the SOUND platform.
- **SOUND Connection Service (SCS)**: The SCS is essentially the SOUND version of IBR. It allows secure connects to be established between members of the Community of Trust, if their reputation is in good standing.
- **SOUND Administration Console (SAC):** A console to allow an administrator to manage the SOUND system, viewing reputations, IBR connections and policies.
- **SOUND Sensor Framework (SSF):** A framework to integrate sensors. The SSF can receive alerts via syslog or parse snort logs. It is extensible to allow the addition of new log formats and sensors in the easiest possible way.

4.2.2 Communities of Trust (CoT)

Participation in a CoT carries an agreement to adhere to a common set of rules. Community members expect others to behave predictably, and punish those that act outside of acceptable boundaries. SOUND requires that CoT members have secure network connections to any other community member they choose to communicate with. Two connection types power SOUND CoT communications: *a priori* and *introduced* connections. The *a priori* connections are like those that might exist within a business group, a military command structure, or a similar real-world community. They represent predefined trusted relationships that SOUND uses to introduce

members to other members across the community, when the two parties do not have an established trust relationship. Any SOUND CoT member may serve as an *introducer*, using *a priori* connections to other community members and introducers to build an introduction path. When all of the introducers along the path agree that the two members may communicate, the members build a protected *introduced* connection with which to transmit data.

Integrity and trust of all community members—introducers included—in a CoT is essential to its operation. To guarantee this level of trust, SOUND promotes accountability by applying an audit function to all nodes. If the members all operate as expected—for any input the expected output and only that output is produced—then the community considers them trustworthy and continues to allow them to participate in the CoT. Once introduced and connected, the members regularly rate the communication and report back through the introduction chain until the interaction concludes. While normal behavior will result in a positive rating, any questionable, suspicious, or blatantly inappropriate behavior will result in a poor rating. This simple model works like human communities where one's past behavior and performance directly impact one's reputation and therefore one's ability to have future community interactions. Just as a good reputation leads to more interactions in human societies, good reputation leads to being allowed to make future connections in a SOUND CoT.

The implications of this simple model and how it operates are powerful. A CoT member (i.e. a human actor, an application, or a host) can either be introduced and thus be an insider, or cannot be introduced and therefore cannot communicate with other CoT members. SOUND records the history of communication ratings as reputation against member identities and uses policies to determine what action, if any, it should take relative to misbehavior. SOUND tolerates minor misbehavior because it may represent a transient user error or something that in isolation is innocuous, events that analysts often interpret as "false positives." However, continued "minor infractions" would eventually result in punitive action. This means a consistent treatment of misbehavior is applied across a CoT up to and including a complete community access revocation if sufficient bad behavior is attributed to a member. A badly behaving insider effectively becomes an outsider and can no longer participate in the community, protecting the CoT operational integrity and all of its members.

4.2.3 Accountability and Monitoring with Mutual Suspicion

In order to provide accountability, we investigated the use of AVM, but discarded this for the reasons outlined below. We created a sensor framework and integrated a number of sensors into SOUND to provide monitoring of community members. Past research suggested that, to protect community assets, communication should only be allowed if the participants could hold each other accountable for their behavior.

4.2.3.1 AVM

Our initial accountability approach was based upon research for generating evidence of misbehavior using AVMs. AVMs require that each CoT host maintain a *tamper-evident log* of local actions and be associated with a set of *witnesses* that audit the host's log. One of the primary challenges for applying AVM to SOUND was that AVMs detected misbehavior by deterministically replaying a host's log using a reference implementation, requiring that the full

behavior be known. We intended for SOUND to extend AVM to verify behavior against a coarse abstraction of expected communications patterns.

The issue we discovered using AVM this way is that it does nothing to protect against defects in the software itself. Most of the exploits in use today are utilizing flaws in the software attacked, not necessarily making changes to that software. The use of ROP gadgets is an example of this. We realized that AVM would not detect this because the flaw would still exist in the gold copy of the software AVM was comparing the running copy against.

4.2.3.2 Tit for Tat

In order to provide a protection model which prevents our reputation system from being used as an attack surface, we utilized the "tit-for-tat" model developed for IBR. When a node reports an attack, both nodes reputation, from the point of view of the other, are reduced. What this means is that if one node attacks another by falsely reporting an attack, eventually the attacking node will no longer be able to communicate with the victim node, and no further false reports can be created. This does not mean the reputation from the perspective of the whole community is degraded to the point of exclusion, so the attacker would need to choose a new target, therefore lower its reputation further in the eyes of the whole community until it is cut off. In this way, we are able to pinpoint the true attacker.

In order for an adversary to use the SOUND reputation system to deny service, the adversary would have to compromise a large number of hosts from a diverse portion of the introducer network. This forces the Adversary to raise the bar and work harder to be able to attack the system.

4.2.3.3 Sensor Integration

SOUND is a system which relies on inputs from sensors. The goal of the SOUND project was not to build new sensors, though we built a few for testing and demo purposes, but rather to integrate existing sensors. During the course of the project, we integrated a number of COTS/GOTS sensors, described in this section.

We integrated with the following sensors:

- **Network Intrusion Detection Sensor –** We integrated with Snort, which is an industry standard Network Intrusion Detection Sensor (NIDS), which is available open source. The snort sensor can detect a wide variety of intrusions through its very extensible ruleset, for which rules are regularly updated. For our purposes, we ran snort separately from SOUND and built a tool which ingested the snort logs. For the demonstrations we did at PACOM, in the NSWC lab and LSD-41 lab, we kept the ruleset very small to detect only port scans and ftp login failures. The reason for this was because the larger rulesets detected so much we could not necessarily control for false alarms or unexpected alerts, which made deterministic demos difficult.
- **ICS Sensor –** We integrated with Sophia, which is a commercial NIDS designed specifically for Industrial Control Systems (ICS) from the company NexDefense, which the Navy was considering for use on shipboard systems. NexDefense was willing to provide us with a complimentary copy of Sophia so we could integrate with it and show how SOUND and Sophia could work together. We did this integration via Syslog. SOUND was able to receive

syslog alerts from Sophia and report them to the reputation system. This was shown in the demonstrations we did in the NSWC lab.

- **Firmware Intrusion Detection Sensors –** We used a proprietary technology developed under the DARPA CRASH program which is able to monitor and alert when issues with the firmware arise. We demonstrated this integration in the LSD-41 lab, by having this sensor detect changes to the firmware of a Programmable Logic Controller (PLC) network card.
- **Hardware Bus Sensor –** We used a proprietary technology hardware sensor which monitors communications between devices. For demonstrations in both the LSD-41 lab and the NSWC lab, we were able to show that SOUND could ingest the hardware bus sensors detection of a logic update and isolate the PLC.

4.2.3.4 SOUND Identity & SOUND Wrapper

Another method we experimented with to assign attribution was in attempting to authenticate and identify and assign reputation to the software in use. We did initial experiments with this using the SOUND Wrapper. SOUND assigns a unique IP address to all identities, and when that user or application authenticated itself, IBR would bind that IP address to the virtual interface it was proxying for. In this way, each individual identity would have its own IBR connections and underlying IPSec tunnels. The SOUND Wrapper ensures that a legacy application will use only the assigned IP address.

The goal in identifying an application is that when you saw bad behavior coming from the user of an application you could lower the reputation of the application (e.g. Firefox) itself a small amount, allowing over time to realize which applications may have built in flaws, even if you do not know what the flaw itself is.

In order to apply this to an application, the application itself would have to authenticate when it started up. We realized that this creates issues with respect to how to authenticate software – having software incorporate its credentials is not secure. In order to test our theories, we created a wrapper – a shell script that you would use to launch the application. The shell script would replace the standard networking libraries with its own libraries. These wrapper libraries would authenticate the application and bind all the applications communications to come from the assigned IP address for that SOUND identity. This allowed us to track the application itself.

The issues we ran into were that using this method we were limited to a single identity being bound to a connection – either the application or the user, since only one IP address could be used. Since this particular piece was not of interest to our specific transition partners, we did not continue the research further. Future research would include determining how to bundle identities for all the applications and pieces of code which were in use by the user, so the behavior seen coming from them would affect the reputation of that code.

4.2.4 Pillars of the Community: Establishing a differentially reliable decentralized trusted environment

4.2.4.1 Reputations

The SOUND Trust Service relies on the SOUND Policy Service and SOUND Reputation Service to determine whether it should trust a member of a community. SOUND uses Sensors,

Trust and Reputation to provide defense in depth for networks. It is a network defender that receives sensor observations of activities performed by and against network-wide identities and aggregates and evaluates them to modify the "reputation" of each identity. Bad behavior by an identity (typically a host or user) reduces its reputation to a degree associated with the actual activity sensed. Individual low-danger activities can add up to significant lowering of reputation and eventual severing of the offending identity from the network. High-danger activities can result in immediate disconnection. SOUND provides broad resilience to attacks because what is detected is behaviors, not (e.g.) virus signatures.

SOUND Trust provides a simple sensor platform which supports easy addition of new sensors and new "suspicious activities," and many ways to adjust its evaluation of behavior events, otherwise known as "impressions."

One benefit we discovered to the addition of reputation for detection of attack is that of relieving alarm fatigue. Network operators or military operators often see hundreds or thousands of alerts a day from various sensors. It's easy to adjust to constantly seeing streams of alerts and no longer respond as quickly or necessarily notice critical alerts in a timely manner. SOUND Reputation helps combat this by adding up the many small alerts into a quantifiable reputation which lets the operator know when a true issue has occurred. The operator can then dig into the many alerts which made up the reputation to determine the cause of the problem.

4.2.4.2 IBR / IBR Modifications

As an extension of standard IBR, SOUND IBR integrates standard IBR's introduction-based connection management scheme with SOUND Platform services.

Figure 10 illustrates how SOUND IBR fits within the SOUND context.

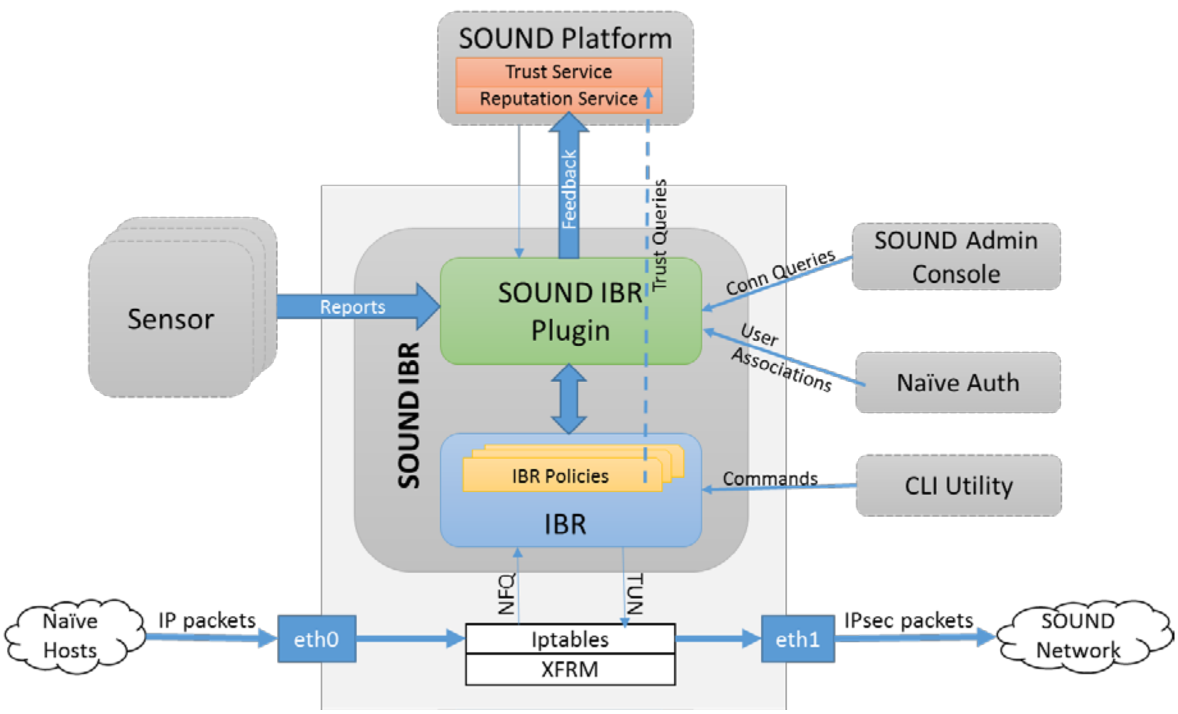


Figure 10: SOUND Architecture with IBR Modifications

As with standard IBR, SOUND IBR serves as a proxy between naïve hosts and the rest of the SOUND network. When a naïve host attempts to connect to another host across the network, SOUND IBR requires that a connection be established to the target. Connections are requested via introduction, which are subject to policy decisions by all nodes involved. IBR enforces connections by manipulating Linux kernel iptables rules and XFRM policies. For more details on standard IBR, see [F+11].

SOUND IBR extends standard IBR chiefly by means of the SOUND IBR Plugin component. The SOUND IBR Plugin connects standard IBR with the various SOUND Platform services (Trust Service, Reputation Service, Naïve Authentication), sensors, and SOUND Admin Console.

Through the SOUND IBR Plugin, IBR policies query the SOUND Trust Service whenever making trust decisions for requesting/offering/accepting introductions. Thus, IBR benefits from the Reputation Service's community-wide perspective as well as its more sophisticated reputation model. Additionally, the IBR policies benefit by being able to incorporate reputations of users as well as hosts when making trust decisions.

In the event that the Trust/Reputation Service becomes unavailable, SOUND IBR provides availability by falling back to using IBR's traditional policy scheme whereby each node continues to make local trust decisions.

SOUND IBR interacts with many SOUND components. The SOUND IBR Plugin has an interface for receiving reports from sensors. It also interfaces with the Reputation Service by forwarding reports to it and receiving reputation updates from it. The Plugin provides on-demand IBR connection data to the SOUND Admin Console. Also, it accepts user association data from Naïve Auth. Finally, IBR also offers a command-line interface used primarily for managing a priori connections.

4.2.5 Formal Specification and Model-based Programming

4.2.5.1 Simple Unified Policy Programming Language (SUPPL)

Simple Unified Policy Programming Language (SUPPL) (Dockins & Tolmach, 2014) is a language for security policies that aims to (1) be simple, expressive, and easily debugged, and (2) support the construction of coherent policies by combination of simpler policies. We start with the premise that a declarative programming language—Prolog—is close to a natural fit for security policies. We then add type and instantiation mode declarations (stating which variables should be bound when a predicate is called, and which are to be bound as a by-product of execution) to catch more errors at compile time and to support compiler optimizations. They add an imperative syntax for conditionally branching event handlers that eliminates some non-logical Prolog constructs (such as "cut") with which even experienced Prolog programmers often shoot themselves in the foot. To make programs easier to debug, reason about, and optimize, we segregate all effects into table updates that can happen only after all handlers for an event have executed. An example of SUPPL code is shown in Figure 11.

A SUPPL program maps exogenous events to exogenous actions. Every handler that matches an event runs against that event. A *coherent* policy handles every event with exactly one set of actions. Unfortunately, it is easy to combine several individually coherent policies to create an

incoherent one. For example, a combined policy might both allow a user access to a resource, because he/she is a superuser (policy 1), and deny access to that resource because he/she is not explicitly authorized to access that resource (policy 2). A human told to combine the superuser-list-based and access-table-based policies would likely assume that superuser status trumps the access table information in the combined policy, and may therefore forget that this constraint must be explicit in the combined policy program. Rather than rank-ordering event-handling rules or actions to resolve such conflicts dynamically (which can have wide-ranging unanticipated effects), SUPPL provides a static analysis to detect potential conflicts, so that the programmer can debug the policy. First, the analysis finds pairs of control-flow paths initiated by the same event types that lead to incompatible actions. For each pair, it defines a first-order logical formula for the conditions that would have to hold for both actions to be derived from the same event. It then submits the formula to an external SMT solver to attempt to prove that the combination of the two paths is unsatisfiable—that is, the two incompatible actions cannot be produced by any single event. If unsatisfiability can't be proven, the pair is marked as a potential conflict. False positives are possible, but have not been a problem in tests to date.

```
primitive type person.
primitive type scanner.
primitive type location.
primitive type door.

data verdict ::= deny
        | open_door (door)
        | dispatch_security (location).
event open_door_request (person, scanner).
action issue_verdict(verdict).

primitive function scan_door(scanner)
        yields door.
primitive function scan_loc(scanner)
        yields location.
primitive function scan_gates(scanner)
        yields location.

primitive predicate
        authorized_loc(person in, location in).

handle open_door_request(?P, ?S) =>
   query
   | authorized_loc(P, scan_loc(S)) =>
     query
     | authorized_loc(P, scan_gates(S)) =>
       issue_verdict(open_door(scan_gates(S)));
     | _ => issue_verdict(deny);
     end;
   | _ =>
     issue_verdict(
       dispatch_security(scan_loc(S)));
     end;
   end;
```

Types must be declared to facilitate type-checking

Special types: composite data, events (which initiate SUPPL execution), and actions (which are executed outside of SUPPL).

Function type signatures allow type checking.

Predicate type signature also includes instantiation mode annotations. "`in`" here means that `authorized_loc` requires both arguments as inputs.

Imperative syntax for event handlers. `?X` indicates a variable binding.

Figure 11: Sample SUPPL Code

The tested implementation of SUPPL targets a Prolog interpreter implemented in Java, which has built-in support for calling Java methods. The aim is to make it easy to integrate SUPPL-defined policies into existing Java applications, or, with modest changes, other languages that have good foreign function interfaces to some Prolog implementation.

4.3 Tests

4.3.1 Single board SOUND Proxy

In order to demonstrate that SOUND could run on a small device which would be easily deployable, we attempted to install SOUND on a single board computer called a PandaBoard. The PandaBoard is similar to a Raspberry Pi, but has a more updated version of the ARM processor, making it easier to find the libraries which SOUND relies on compiled for it.

Our efforts showed that a SOUND proxy could run effectively on a low-power single-board computer. We were able to run a proxy and stream video over it with no loss of quality.

4.3.2 Blue-forces Mission-Oriented Command Center (BMOC)

As we continued forward with the SOUND project, it became obvious that we would need an application we could use to demonstrate and test how SOUND worked, to which we would be able to build in flaws and needed features. As a result, we created the Blue-forces Mission-Oriented Command Center (BMOC). BMOC was designed to be able to show how a typical application might be able to utilize the powers of SOUND. An example of the BMOC interface can be seen in Figure 12.

Figure 12: BMOC Interface

The BMOC functions include:

- **BMOC:** A Java web application running with embedded Tomcat, providing functionality for the UAV video display, UAV positional display, chat messaging, and Call For Fires (SAFE) requests. The web application transmits and receives data via a Representational State Transfer (REST) interface, and serves static Hypertext Markup Language (HTML) and JavaScript files to a requesting web-based browser client. The RESTful interface is implemented with Jersey *Service classes. BMOC has several functionality modules, including: UAV, Call For Fires, Chat, Admin, Debug, and Attack. The SOUND Software Design Document (SDD) further describes each module.
- **Web Browser:** BMOC requires the Firefox web browser with a custom SOUND extension to access application resources. All BMOC pages are static HTML. Periodic data requests are made to the BMOC server via Asynchronous JavaScript and XML (AJAX).
- **UAV:** The UAV simulators are separate java processes, each with unique initial configuration for location, destination, altitude, airspeed, and heading. Every second, each simulator calculates its new location based on its configuration and sends its location information to the BMOC web application via a RESTful call. Each simulator also has a RESTful listener to enable the External Spoofing Attack and to reset the UAV to its initial settings.
- **SOUND Data Service:** SDS retains and serves the messages for the Chat module.
- **VLC Server:** BMOC uses the open source VLC video media server to stream the video component of UAV module. A second VLC server serves to deliver a competing video stream to interrupt the UAV stream as part of the DoS Attack.

4.3.3 Red Team

We performed two Red Team Assessments on SOUND during the program. The first was a table-top exercise where the Red Team looked over the design and code and made suggestions. The second was a full Red Team exercise where we set up a live network for the Red Team to attempt to exploit the SOUND and NPD integration which we later deployed in the LSD-41 lab.

The Red Team was able to find a number of issues which were addressed for the LSD-41 demonstration, and a number that should be addressed if SOUND is deployed in a production environment such as hardening of the server bindings, tighter privileges, input sanitization, and access permissions. These issues are listed in Table 2 of the VDD.

Overall, the Red Team felt the design of the system was solid and did not find any opportunities for Man-In-The-Middle (MITM) attacks or the introduction of malicious introducers into the system. They did not feel the integration of NPD and SOUND increased the attack surface. Once hardening is completed, they felt the sensors would be the weakest spot in the system, so sensor security should be a priority in a deployed SOUND system.

4.4 Demonstrations

During the life of the project, we performed a number of increasingly sophisticated demonstrations, tests and experiments for transition partners, including SOCOM, NAVSEA, PACOM, and DISA. This section will outline the major ones.

4.4.1 Chat Server

At the November 2012 PI Meeting, we did an initial demonstration of the SOUND integrated technologies using a chat server. For this demo, we incorporated a current implementation of IBR (spiral 7) and the notion of AVM through logging and replay mechanisms. We extended the Apache Mina/Vysper XMPP chat server for logging and replay mechanisms as well as added a trivial data exfiltration exploit. We wrote a plugin for the Ignite Spark XMPP chat client to allow clients to ‘report misbehavior’ about other clients. In short the two scenarios were focused around mitigating data exfiltration and mitigating chat messages sent by potentially malicious entities.

We used a very naive logging and replay mechanism that resembles AVM in spirit. On the chat server, each time a message is received, a copy of the message is forwarded to a reference chat server (the ‘gold version’). That is, we assume the ‘gold version’ has not been corrupted and cannot be corrupted by receiving normal XMPP messages. Similarly, whenever the chat server sends a message, a copy of the sent message is forwarded to the reference server. In an infinite loop, the reference server listens for each type of message. Messages received by the original chat server are passed along as regular messages (i.e. the reference server sees these messages as ones from clients). Messages sent by the original chat server are put into a queue. Whenever the reference server produces an output message, the AVM replay mechanism checks that the output message matches one on the queue of messages. When the messages don’t match, AVM notifies IBR of the bad behavior of the original chat server.

We installed IBR on 7 virtual machines running Fedora. The introducer was configured with a priori connections to each of the other VMs, so the only way that the clients were able to connect to the chat server was via IBR introductions. In this way, we were able to disconnect clients from the chat server when it was deemed to be misbehaving by notifying the IBR introducer of its misbehavior.

We included two attack scenarios. The first involved the chat server being compromised and exfiltrating messages to a third party. The second involved a malicious user sending fake messages, with user identified misbehavior being reported, and ultimately disconnecting the server.

4.4.1.1 Outcomes

There are two large realizations which came out of this demonstration. The first was that there needs to be some story for resiliency. Stopping the world won’t suffice. The second realization was that the reason for only being able to disconnect the chat server, and not just the attacker, was that we only have the notions of connection and identity at the IP level. As a result, if you don’t have a direct connection to someone, you don’t really know their identity in the scheme of

IBR. As a result, and after considering other web applications, the notions of identity and reputation have to exist at the application level.

4.4.2 SOUND Router

As we envisioned that ultimately deployments of SOUND would be more performant and secure if the encryption and IBR tunnels could be performed at the hardware level on a SAFE processor (developed under the DARPA CRASH program). This section summarizes our work; more details can be found in [K+14].

4.4.2.1 Concept

Denial of service (DoS) and distributed denial of service (DDoS) attacks exploit a basic design principle of the internet; any machine can send a packet to any other. The IPsec protocol makes it possible to filter out unwanted packets, but only at the destination, on a per-packet basis, allowing an attacker to saturate the targeted machine's ports with unwanted packets. RotoRouter (Kwon, et al., 2014) is a novel protocol and router architecture designed to prevent DoS attacks by filtering unwanted packets on the router itself, so that they never reach the targeted endpoint. To verify that a packet is "wanted" by the destination, the router

1. Asks the destination if it wants to receive packets from the source, and forwards packets only if the answer is yes; and
2. Requires the source to sign each packet to verify its origin and integrity.

Figure 13 shows a picture of the RotoRouter prototype.

Figure 13: RotoRouter Prototype

4.4.2.2 Protocol

When the router first sees the requested connection (source IP, destination IP, and port #) there is some one-time (as opposed to per packet) set up:

1. The router computes a connection ID from the source IP, destination IP, and a random bit string such that (a) neither IP can be read off of the ID, and (b) it is difficult for an attacker to create a valid ID.
2. The router asks the destination if it wants to receive packets from this source. If the answer is “no”, future packets from the rejected source can be identified and dropped just by looking up the connection ID in a table and checking the result’s “allowed” field.
3. The source generates a private/public key pair, and sends the public key to the destination.

A router handling an existing connection for the first time can query the destination for the public key, which it will need to authenticate packets.

For each packet:

1. The source computes a hash (message digest) of the connection ID, payload, and several header fields.
2. The router decrypts the message digest and recomputes it against the packet to verify that the packet came from the connection’s identified sender and that its contents have not been altered.
3. If the packet is verified, it is forwarded. Otherwise it is dropped.

4.4.2.3 Implementation

The protocol was tested in an implementation aboard a NetFPGA-10G platform, an FPGA board that supports prototyping and experimenting with network protocols and hardware.

To keep high throughput, a few of RotoRouter’s more expensive/frequent tasks required special attention. A flow table maps connection IDs to information about the connection—whether the connection is allowed, the RSA public key used for authentication, etc. Because every packet requires a lookup in this table, it is implemented as a dynamic multi-hash cache, a species of content-addressable memory. RotoRouter also dedicates hardware cores to two of the most expensive steps in the protocol: SHA-1 hashing and modular exponentiation required for packet authentication. RotoRouter uses about five times the resources of a comparison open source IPv4 router, with most of the extra due to these two modules. However, the overall throughput is a respectable 8 Gbps. Figure 14 shows resource usage and throughput for the RotoRouter.

Module	Area		Clock (MHz)
	LUTs	BRAMs	
Crossbar w/ Buffers	8249	16	300
Flow Table	38	74	350
Processor	26985	52	200
SHA-1 Module	4x1005	0	125
Mod-Exp	73591	0	200
RotoRouter	112883	142	125
IPv4 Router	22523	35	150
Total available	149760	324	-

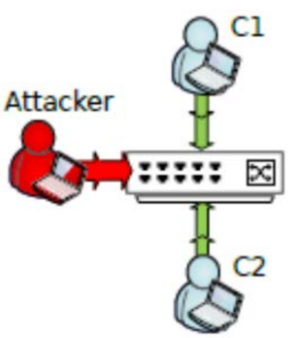
	Crossbar	Flow Table	SHA-1	Mod-Exp
Clock Speed (MHz)	300	350	125	200
Individual Throughput (Gpbs)	19.2	515	4 x 0.8	4 x 1.2
Effective Throughput @ 125 MHz (Gbps)	8	184	3.2	4.8

Figure 14: Resource Usage and Throughput

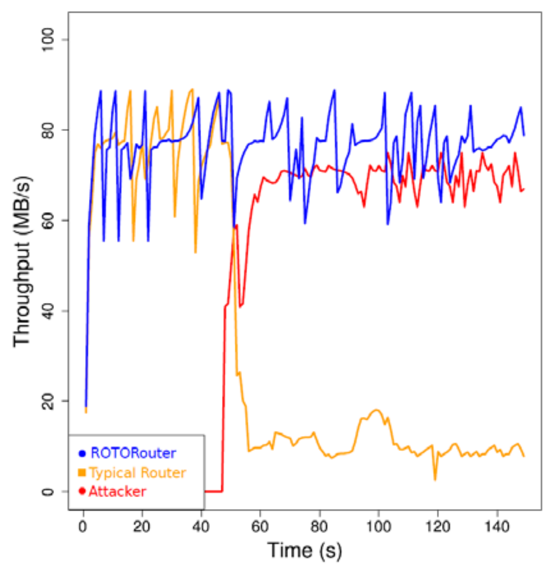
4.4.2.4 Simulated attack tests

RotoRouter was tested against a small simulated attack: two machines on an isolated network are video-chatting, while a third, malicious machine attempts to flood the network with junk messages.

With the RotoRouter's special features disabled (so running as a vanilla IPv4 router), the throughput of legitimate data plummets immediately upon attack, but with RotoRouter enabled, there is no measurable drop at all. Figure 15 illustrates the attack test experimental setup (a) and goodput measured (b).



(a) Experimental Setup



(b) Throughput of ROTORouter and Typical Router Under DoS Attack

Figure 15: Attack Test

4.4.2.5 Avoiding DoS attacks on the router itself

Because there is costly set-up associated with each new connection, a RotoRouter instance is vulnerable to an attacker sending numerous bogus new connections. To mitigate this, the invalid request rate on each input port is tracked. When this rate exceeds a threshold, the port is considered to be under attack and is shut down.

The throughput, resource usage, and basic filtering behavior of a RotoRouter instance have been demonstrated, but larger-scale tests remain, to (e.g.) understand the dynamics of connection validation traffic and optimize such parameters as the number of unverified packets to let through at the start of a connection.

The cost of RotoRouter's extra validation steps is significant, but not prohibitive, given current technologies, and promises to be effective at preventing DoS attacks.

4.4.3 Persistent Insider Scenario Demonstration

For the January 2014 PI Meeting, we demonstrated how SOUND would protect against a persistent insider. We utilized our BMOC application to show the simulated path of a UAV. BMOC incorporated a chat server and a GPS tracker. We showed a situation where a flaw in BMOC allowed an insider Ivan to give himself permissions to access to information he shouldn't have had. The demonstration first showed what Ivan could do if SOUND was not in effect. Ivan is able to utilize the bug to gain access to his commander's chat history. When SOUND is enabled, we allow Ivan to still exploit the bug to change his permissions within the application, since the premise is that we don't know about this bug in the application, but the SOUND Data Service recognizes that Ivan is not allowed to have access to the data. The Data Service intercepts the request, detects and escalation in Ivan's privilege and prevents Ivan from receiving the data. In addition, the attempt to access the data is logged. The policy in effect for SOUND is that of '2 strikes and you are out' and once Ivan attempts to access the data again – he is logged off the BMOC system entirely.

4.4.3.1 Outcomes

- We were able to demonstrate SOUND in a physical networked environment by having 4 separate laptops for the SOUND Services, and the BMOC Server and client.
- Equipment included four machines, including two end-user client hosts, one application server, and one that hosts the SOUND Core components.
- Demonstrated how a combination of sensor outputs could be combined to make a policy decision about whether to remove a user from the Community of Trust (CoT)

4.4.4 PACOM

In August and November of 2014, we went out to PACOM in Hawaii to demonstrate SOUND on a realistic operational network in the SPAWAR Systems Center Pacific Facility. We presented a Humanitarian Aid / Disaster Relief (HADR) scenario involving military forces required to work with untrusted Non-Government Organizations (NGOs), who attempt to use the network connections they are given for video communications to attack the military networks. We had 13 SOUND nodes in this scenario, Figure 16 shows the network setup for this demonstration. Four

different groups were communicating on the network: 3 military groups and an NGO. We demonstrated their ongoing communications by streaming video.

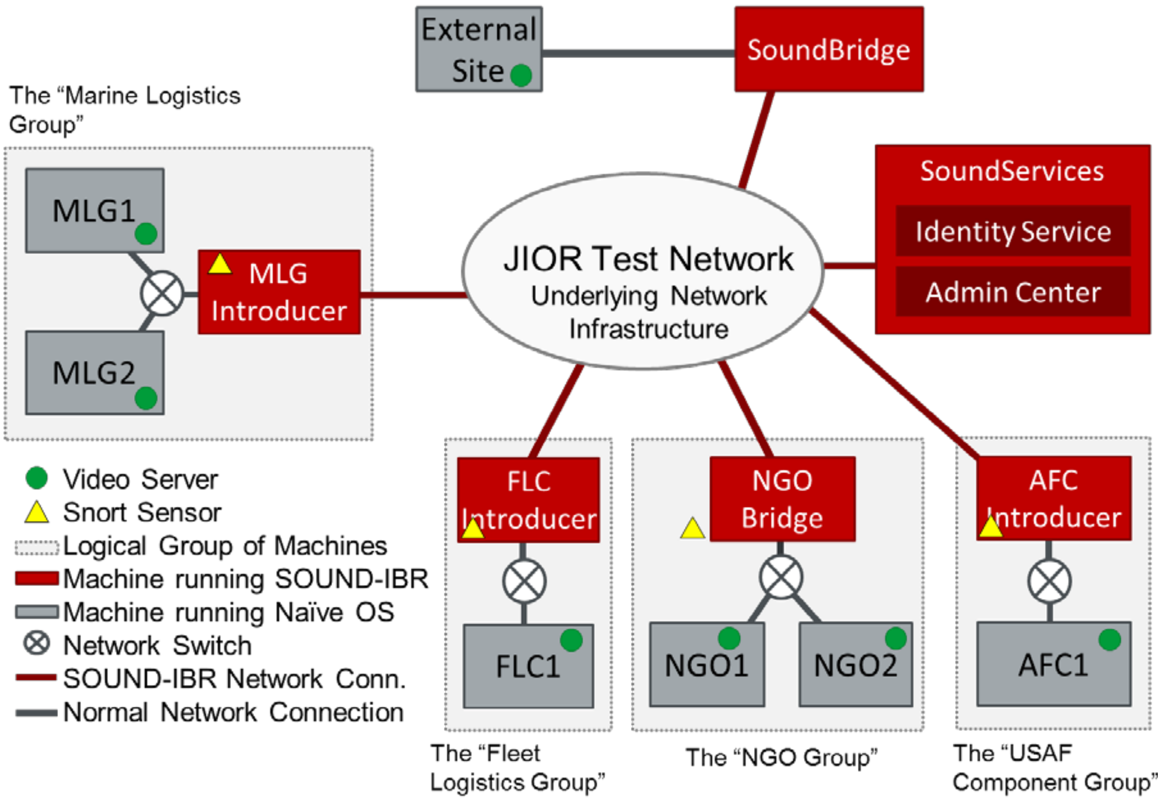


Figure 16: Network Configuration for PACOM Demonstration

We showed how if a bad actor in the NGO facility attacked one of the military nodes, SOUND would isolate it from communicating with the victim, but not remove it from the Community of Trust entirely. When the bad actor attacked a second military node, SOUND removed the bad actor from the community of interest. The attacker then attempted to move to another node, where he still did not have access, since his account was no longer allowed in the CoT, regardless of where he logged in. We then showed that if another user in the NGO were to attack one of the military nodes, based on policy, SOUND would remove the entire NGO from the CoT.

4.4.4.1 Outcomes

- Demonstrated SOUND's capability to protect differential networks
- Showed SOUNDs integration with the Snort COTS sensor
- Demonstrated SOUND to a large number of interested groups from across the military.
- Generated interest from PACOM in potentially including SOUND in an exercise, which did not later materialize.

4.4.5 SOUND+NPD Integration

We were asked by DARPA to look into whether integrating with any of our sister projects on the MRC program would be beneficial. After discussions with a fellow performer, Applied Communications Sciences (ACS), we determined it would be worthwhile to explore an integration between SOUND and their Network Path Diversity (NPD) technology. This worked out well as the combination worked very well for the Navy demonstrations we did.

NPD provides path diversity in a network by routing traffic through other NPD nodes as an overlay on the existing network, for which you may not have control over routing. NPD can also send packets along multiple paths, to ensure greater availability of the data.

SOUND provides insider threat protection, enclave protection to NPD (which is not otherwise hardened), supports health assessments of NPD's intermediaries and can influence/help control NPD's list of available intermediaries based on reputation.

NPD provides defensive maneuver capabilities by giving us the ability to route traffic through NPD intermediaries. NPD can also act as a network traffic sensor, letting us consider the reliability of a network path when determining the reputation of an intermediary to use.

Together, they can defend against malicious behavior inside networks by having SOUND monitor reputations, log misbehaviors and isolate serious attacks. Additional benefits include the ability to use more costlier (in computing terms) sensors, which would be too cumbersome to run locally. NPD can divert traffic past the more costly sensor while still sending it along the normal path. SOUND+NPD also protects against attacks from the outside by having NPD fight through network degradation, not matter what the cause is, and SOUND wraps ad hock communities of interest inside enclaves to restrict access and protect enclave participants. SOUND+NPD empowers the defense of the network by giving the operator the ability to reroute traffic away from suspicious nodes.

Integration

Integration between the two technologies was done loosely, by installing both technologies on the same box and deconflicting their iptables rules. NPD sits logically "behind" the SOUND proxy, somewhat as if it was a separate box connected behind it, functioning as a bump in the wire. Figure 17 shows the architecture of the SOUND and NPD integration.

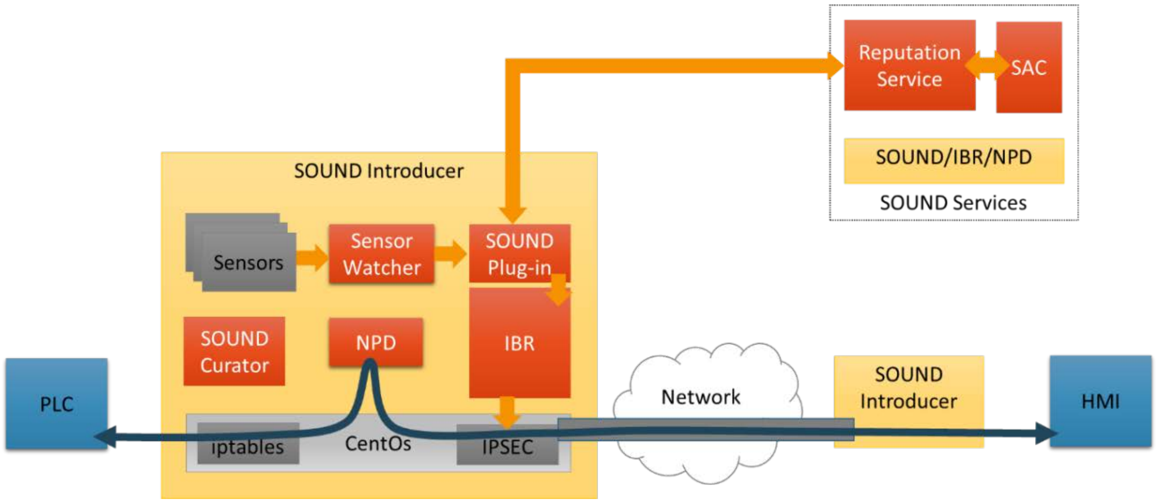


Figure 17: Architecture for SOUND+NPD

We demonstrated the SOUND+NPD integration to DARPA.

4.4.5.1 Outcomes

- The exploration into how SOUND+NPD could work together let us bring a more complete capability to discussions with NAVSEA NSWC
- The combination of the two technologies allowed us to operate on the shipboard ICS networks with no modification to the networks, which is what made the NSWC lab demonstration possible.

4.4.6 NSWC Research Lab

We were invited to the NAVSEA Navy Surface Warfare Center (NSWC) Philadelphia to show how SOUND, in conjunction with our sister project from Applied Communications Sciences (ACS) Network Path Diversity (NPD) could be used to protect Navy ICS shipboard systems. NSWC had a research lab for experimenting with potential products they might want to use.

When we first arrived in the NSWC lab, we learned two critical things. SOUND+NPD needed to be transparent to the Navy systems and not require any changes to network configuration on their side, and second, that the Navy uses a significant amount of multicast and we needed to be able to handle this traffic. We were able to address the first issue by spoofing ARP so that the Navy machines saw the proxy as the machine they were talking to. With the help of NPD, we were able to address the second by tunneling the multicast traffic through the network via NPD.

Figure 18 shows the network setup for this demonstration. We integrated SOUND with Sophia (more info in the Sensor Integration section) and showed how SOUND could be used to protect a network modeled after the Mohawk class Coast Guard cutter. SOUND protected and isolated the Programmable Logic Controller (PLC) from an attack which moved laterally around the network. We ran an nmap scan to emulate a reconnaissance phase, simulated a malware attack by downloading and then changing the PLC ladder logic. We used reputation a little differently in this demonstration by quarantining an infected host and isolating a vulnerable PLC from further attack.

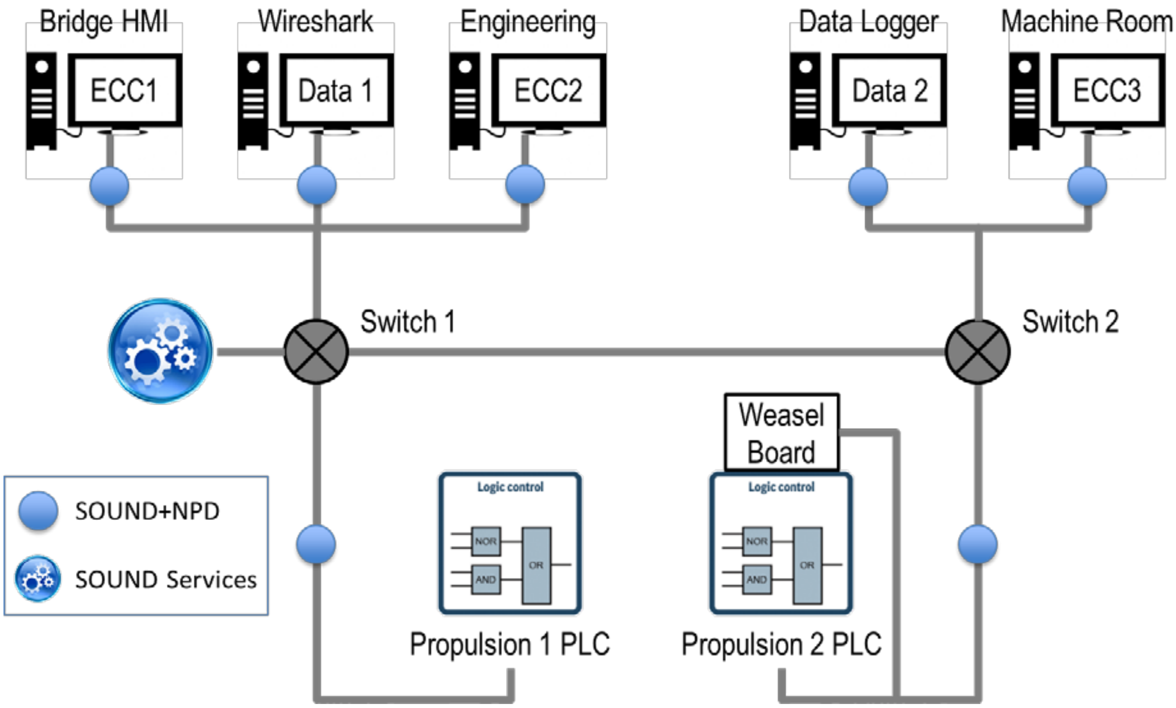


Figure 18: Network Configuration for NSWC Lab Demonstration

Attack scenarios included in NSWC Lab Demonstration:

- Nmap Scan
- Loading a web page on PLC
- Uploading ladder logic from PLC
- Modifying and downloading Ladder logic to PLC

4.4.6.1 Outcomes

- Demonstrated to numerous Navy decisions makers, including a Vice-Admiral.
- Generated interest in a larger deployment on in a Navy ship lab
- Based on this demonstration, NAVSEA agreed that they would sign a Technology Transfer Agreement (TTA) if SOUND could be demonstrated at a larger scale in the LSD-41 class ship lab.

Figure 19 shows a logical layout of the demonstration (left) and outlines the features we demonstrated (right).

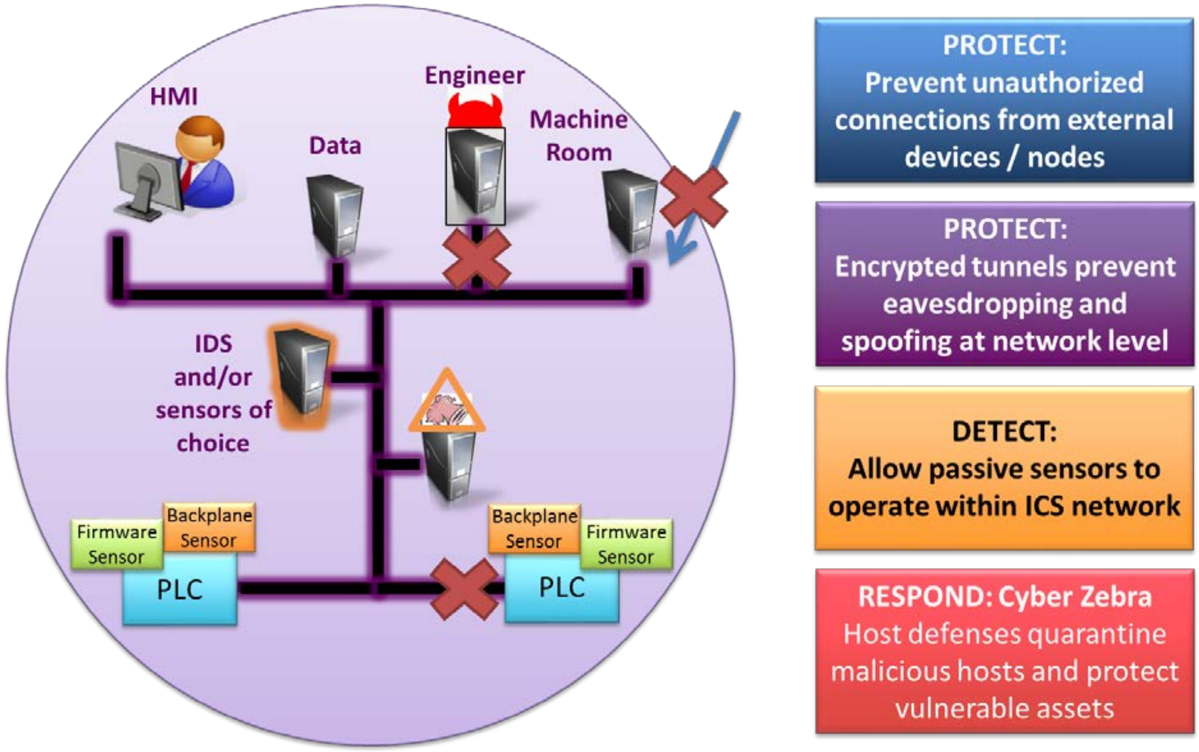


Figure 19: Notional Demonstration Architecture and Features

4.4.7 NSWC LSD-41 lab

NSWC Philadelphia liked our demonstration in the NSWC research lab and wanted to see a demonstration which was more to the scale of a real ship. The LSD-41 was chosen as a ship whose lab was available and was a reasonable enough representation for a viable demo. We faced several challenges with this deployment. First was the scale – the software had not been tested at this scale and we discovered several issues which needed to be dealt with. Second was deployment – maintaining and deploying software to so many systems was tedious and required a software solution to make it tractable. Third was management of the systems – we wrote a curator software to manage the system from an operator standpoint, to be able to know when systems were functioning correctly. Figure 20 shows the network layout of the LSD-41 demonstration.

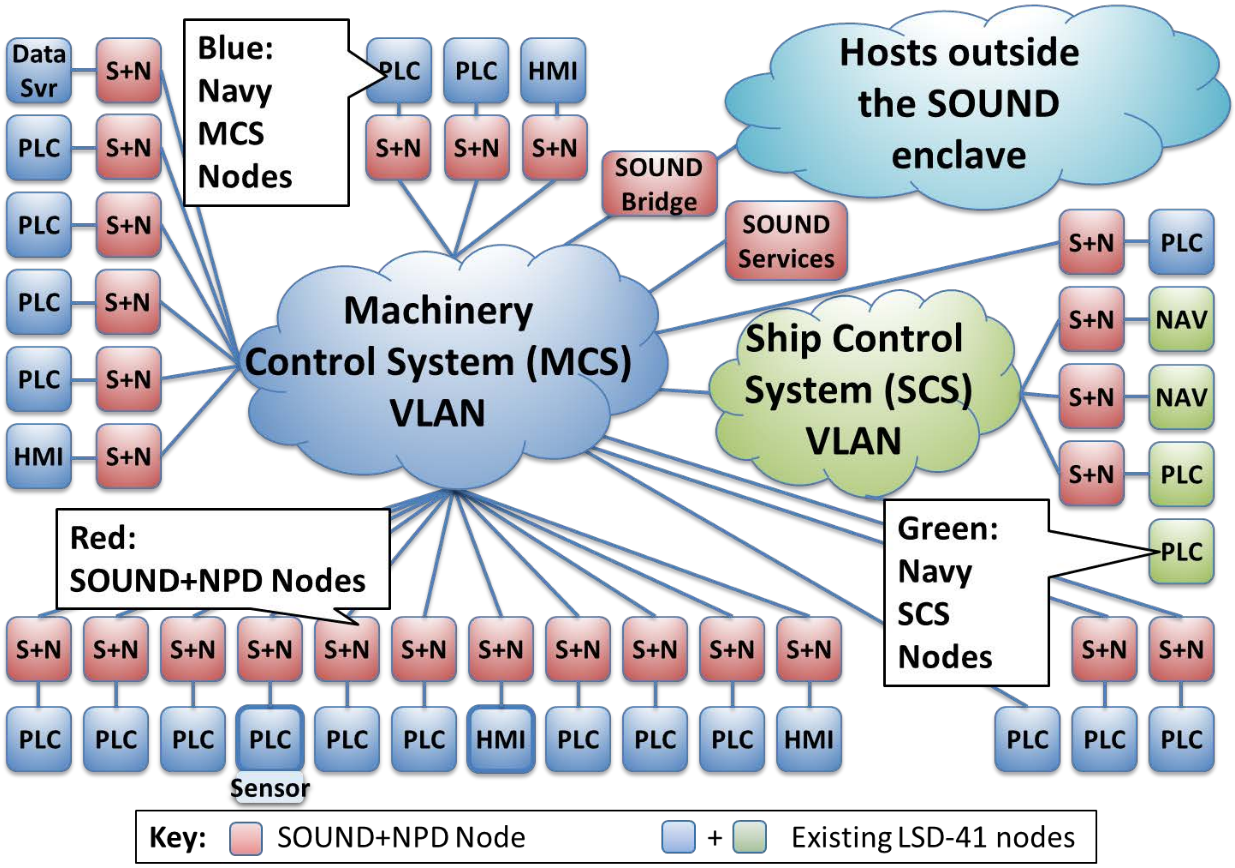


Figure 20: Layout of LSD-41 Demonstration Network

4.4.7.1 Outcomes

- We were able to demonstrate that SOUND+NPD could be deployed on a large scale shipboard ICS network and not require the modification of or interfere with the operation of the shipboard systems.
- We were able to show how SOUND+NPD could integrate with firmware and hardware sensors to detect attacks on the backplane and within the firmware of the PLC network card.
- We demonstrated how SOUND+NPD could isolate a PLC, allowing the ship system to failover to its backup.
- NAVSEA agreed to sign the TTA to transfer the Reputation technology to its cyber situational awareness system.

4.5 Publications

The following publications were published by the SOUND team during the course of the program:

[DT14] Robert Dockins, and Andrew Tolmach. SUPPL: A flexible language for policies. In *Proceedings of the 12th Asian Symposium on Programming Languages and Systems (APLAS 2014)*, Nov 2014.

Abstract:

We present the Simple Unified Policy Programming Language (SUPPL), a domain-neutral language for stating, executing, and analyzing event-condition-action policies. SUPPL uses a novel combination of pure logic programming and disciplined imperative programming features to make it easy for non-expert users to express common policy idioms. The language is strongly typed and moded to allow static detection of common programming errors, and it supports a novel logic-based static analysis that can detect internally inconsistent policies. SUPPL has been implemented as a compiler to Prolog and used to build several network security applications in a Java framework.

[TDT15] Alix Trieu, Robert Dockins, and Andrew Tolmach. Static conflict detection for a policy language. In *Proceedings of Vingt-sixièmes Journées Francophones des Langages Applicatifs (French-speakers' Workshop on Functional Languages) (JFLA 2015)*, Jan 2015.

Abstract:

We present a static control flow analysis used in the Simple Unified Policy Programming Language (SUPPL) compiler to detect internally inconsistent policies. For example, an access control policy can decide to both “allow” and “deny” access for a user; such an inconsistency is called a conflict. Policies in Suppl. follow the Event-Condition-Action paradigm; predicates are used to model conditions and event handlers are written in an imperative way. The analysis is twofold; it first computes a superset of all conflicts by looking for a combination of actions in the event handlers that might violate a user-supplied definition of conflicts. SMT solvers are then used to try to rule out the combinations that cannot possibly be executed. The analysis is formally proven sound in Coq in the sense that no actual conflict will be ruled out by the SMT solvers. Finally, we explain how we try to show the user what causes the conflicts, to make them easier to solve.

[FUR15] Michael Figueroa, Karen Uttecht, and Jothy Rosenberg. A SOUND Approach to Security in Mobile and Cloud-Oriented Environments. In *Proceedings of 2015 IEEE International Symposium on Technologies for Homeland Security,* Boston, 14-16 April 2015.

Abstract:

Ineffective legacy practices have failed to counter contemporary information security and privacy threats. Modern IT operates on large, heterogeneous, distributed sets of computing resources, from small mobile devices to large cloud environments that manage millions of connections and petabytes of data. Protection must often span organizations with varying reliability, trust, policies, and legal restrictions. Centrally managed, host-oriented trust systems are not flexible enough to meet the challenge. New research in distributed and adaptive trust frameworks shows promise to better meet modern needs,

but lab constraints make realistic implementations impractical. This paper describes our experience transitioning technology from the research lab to an operational environment. As our case study, we introduce Safety on Untrusted Network Devices (SOUND), a new platform built from the ground up to protect mobile and cloud network communications against persistent adversaries. Initially based on three founding technologies- Accountable Virtual Machines (AVM), Quantitative Trust Management (QTM), and Introduction-Based Routing (IBR)- our research efforts extended those technologies to develop a more powerful and practical SOUND implementation.

[ADG15] Arthur Azevedo de Amorim, Maxime Dénès, Nick Giannarakis, Cătălin Hriţcu, Benjamin C. Pierce, Antal Spector-Zabusky, and Andrew Tolmach. Micro-Policies: Formally Verified, Tag-Based Security Monitors. In *Proceedings of IEEE Security and Privacy 2015,* San Jose, May 2015.

Abstract:

Recent advances in hardware design have demonstrated mechanisms allowing a wide range of low-level security policies (or micro-policies) to be expressed using rules on metadata tags. We propose a methodology for defining and reasoning about such tag-based reference monitors in terms of a high-level "symbolic machine" and we use this methodology to define and formally verify micro-policies for dynamic sealing, compartmentalization, control-flow integrity, and memory safety, in addition, we show how to use the tagging mechanism to protect its own integrity. For each micro-policy, we prove by refinement that the symbolic machine instantiated with the policy's rules embodies a high-level specification characterizing a useful security property. Last, we show how the symbolic machine itself can be implemented in terms of a hardware rule cache and a software controller.

[K+14] Albert Kwon, et al. "RotoRouter: Router support for endpoint-authorized decentralized traffic filtering to prevent DoS attacks." Field-Programmable Technology (FPT), 2014 International Conference on. IEEE, 2014.

Abstract:

RotoRouter addresses Denial-of-Service (DoS) attacks on networks with a novel protocol and router implementation. Sets of RotoRouters cooperate in detecting and filtering out invalid network traffic before it reaches network endpoints; a new router-enforceable connection protocol queries destination endpoints to authorize traffic flows and uses per-packet digital signatures to distinguish allowed from disallowed connections. A RotoRouter prototype was implemented on a four-port 1000BASE-T NetFPGA-10G platform and supports 1024 simultaneous active connections using 74 BRAMs (less than one quarter of the available NetFPGA-10G BRAMs). It is able to sustain 800 Mbps per port throughputs for 1500B packets with less than 0.3/its latency, even during a DoS attack. With additional logic and memory resources, the required validation and switching operations scale to port speeds in excess of 10 Gbps and links with more than 10,000 active flows.

5.0 CONCLUSIONS

The SOUND project successfully created a technology which was able to create secure Communities of Trust, provide multi-level tracking of reputation for accountability, and isolate actors from the community if their reputation dropped below the policy-defined threshold.

We were able to demonstrate successfully:

- How SOUND could be used to protect infrastructure when communicating with organizations of unknown trustworthiness. Using Reputation and a variety of sensors, SOUND can track users behavior and downgrade their level of trust. Trust can be differential, without having to be all or nothing, so less trustworthy partners can be interacted with and the organization will know that if they show signs of compromise or attacks, they will be removed from the CoT.
- How SOUND could be used to protect Shipboard SCADA systems. We deployed SOUND on a full-scale shipboard network and demonstrated that SOUND could be used to protect equipment and isolate attackers before damage to the ship could be done.
- How SOUND could integrate with a variety of GOTS/COTS sensors. We integrated with a number of different sensors from those available open source, proprietary commercial and developed by government labs.
- SOUND does not require any changes to existing applications or hosts. In our demonstrations in the LSD-41 labs, we were able to install SOUND without having to modify the software, hosts or PLCs running the shipboard network. Our technology can be used with any legacy IP based systems or applications.

5.1 Recommendations

- We recommend that the Navy deploy the SOUND technology onto a ship for testing, with the later intention of putting it into ships throughout their fleet.
- Additionally, SOUND is ideal for use in a traditional office network, and we are working with BAE systems to deploy beta versions to the corporate network, we recommend the government consider deploying SOUND on its IT networks.
- We recommend that the government expand on its use of SOUND Reputation beyond that of basic reputation as deployed in LSD-41 to utilize its full potential in managing Communities of Trust.
- We recommend the military consider that SOUND could be deployed to multiple domains, especially those that employ multiple enclaves or Communities of Trust which need to communicate.

6.0 REFERENCES

[BFK98] Matt Blaze, Joan Feigenbaum, and Angelos D. Keromytis. Keynote: Trust management for public-key infrastructures (position paper). In Bruce Christianson, Bruno Crispo, William S. Harbison, and Michael Roe, editors, Security Protocols Workshop, volume 1550 of Lecture Notes in Computer Science, pages 59–63. Springer, 1998.

[BKL+09] Matt Blaze, Sampath Kannan, Insup Lee, Oleg Sokolsky, Jonathan M. Smith, Angelos D. Keromytis, and Wenke Lee. Dynamic trust management. IEEE Computer, 42(2):44–52, 2009.

[CVW+11] Jian Chang, Krishna K. Venkatasubramanian, Andrew G. West, Sampath Kannan, Boon Thau Loo, Oleg Sokolsky, and Insup Lee. As-trust: A trust quantification scheme for autonomous systems in bgp. In McCune et al. [MBP+11], pages 262–276.

[JHP06] Audun Jøsang, Ross Hayward, and Simon Pope. Trust network analysis with subjective logic. In Proceedings of the 29th Australasian Computer Science Conference – Volume 48, Vladimir Estivill-Castro and Gillian Dobbie (Eds.), Vol. 48, pages 85–94. Australian Computer Society, 2006.

[WAC+09] Andrew G.West, Adam J. Aviv, Jian Chang, Vinayak S. Prabhu, Matt Blaze, Sampath Kannan, Insup Lee, Jonathan M. Smith, and Oleg Sokolsky. Quantm: a quantitative trust management system. In Evangelos P. Markatos and Manuel Costa, editors, EUROSEC, pages 28–35. ACM, 2009.

[BDHU09] Michael Backes, Peter Druschel, Andreas Haeberlen, and Dominique Unruh. CSAR: a practical and provable technique to make randomized systems accountable. In Proceedings of the 16th Annual Network & Distributed System Security Symposium (NDSS'09), Feb 2009.

[Hae09] Andreas Haeberlen. A case for the accountable cloud. In Proceedings of the 3rd ACM SIGOPS International Workshop on Large-Scale Distributed Systems and Middleware (LADIS'09), October 2009.

[HARD09] Andreas Haeberlen, Ioannis Avramopoulos, Jennifer Rexford, and Peter Druschel. NetReview: Detecting when interdomain routing goes wrong. In Proceedings of the 6th Symposium on Networked Systems Design and Implementation (NSDI'09), Apr 2009.

[HARD10] Andreas Haeberlen, Paarijaat Aditya, Rodrigo Rodrigues, and Peter Druschel. Accountable virtual machines. In Proceedings of the 9th USENIX Symposium on Operating Systems Design and Implementation (OSDI'10), October 2010.

[HK09] Andreas Haeberlen and Petr Kuznetsov. The Fault Detection Problem. In Proceedings of the 13th International Conference on Principles of Distributed Systems (OPODIS'09), December 2009.

[HKD07] Andreas Haeberlen, Petr Kuznetsov, and Peter Druschel. PeerReview: Practical accountability for distributed systems. In Proceedings of the 21st ACM

Symposium on Operating Systems Principles (SOSP'07), Oct 2007.

[DT14] Robert Dockins, and Andrew Tolmach. SUPPL: A flexible language for policies. In *Proceedings of the 12th Asian Symposium on Programming Languages and Systems (APLAS 2014)*, Nov 2014.

[TDT15] Alix Trieu, Robert Dockins, and Andrew Tolmach. Static conflict detection for a policy language. In *Proceedings of Vingt-sixièmes Journées Francophones des Langages Applicatifs (French-speakers' Workshop on Functional Languages) (JFLA 2015)*, Jan 2015.

[ADG15] Arthur Azevedo de Amorim, Maxime Dénès, Nick Giannarakis, Cătălin Hriţcu, Benjamin C. Pierce, Antal Spector-Zabusky, and Andrew Tolmach. Micro-Policies: Formally Verified, Tag-Based Security Monitors. In *Proceedings of IEEE Security and Privacy 2015*, San Jose, May 2015.

[K+14] Albert Kwon, et al. "RotoRouter: Router support for endpoint-authorized decentralized traffic filtering to prevent DoS attacks." Field-Programmable Technology (FPT), 2014 International Conference on. IEEE, 2014.

[SS01] Jordi Sabater, and Carles Sierra. "Regret: A reputation model for gregarious societies." Fourth workshop on deception fraud and trust in agent societies. Vol. 70. 2001.

[FUJ15] Michael Figueroa, Karen Uttecht, and Jothy Rosenberg. "A SOUND approach to security in mobile and cloud-oriented environments." *Technologies for Homeland Security (HST), 2015 IEEE International Symposium on*. IEEE, 2015.

[F+11] Gregory Frazier, et al. "Incentivizing responsible networking via introduction-based routing." *International Conference on Trust and Trustworthy Computing*. Springer, Berlin, Heidelberg, 2011.

LIST OF ACRONYMS, ABBREVIATIONS, AND SYMBOLS

ACRONYM	DESCRIPTION
ACM	Association for Computing Machinery
ACS	Applied Communication Sciences
AFRL	Air Force Research Laboratory
AFC	Air Force Component
AJAX	Asynchronous JavaScript and XML
ARM	Advanced RISC Machines
ARP	Address Resolution Protocol
AVM	Accountable Virtual Machine
BGP	Border Gateway Protocol
BMOC	Blue Force Mission-Oriented Command Center
CA	California
CoT	Community of Trust
COTS	Commercial Off-The-Shelf
CRASH	Clean-slate design of Resilient Adaptive Secure Hosts
CV	Compliance Value
CWE	Common Weakness Enumeration
DARPA	Defense Advanced Research Projects Agency
DDoS	Distributed Denial of Service
DISA	Defense Information Systems Agency
DM	Decision Management
DoD	Department of Defense
DoS	Denial of Service
EUROSEC	European Workshop on Systems Security
FLC	Fleet Logistics Center
FPGA	Field-Programmable Gate Array
Gbps	Gigabits per second
GOTS	Government Off-The-Shelf
GPS	Global Positioning System
HADR	Humanitarian Aid/Disaster Relief
HMI	Human-Machine Interface
HTML	Hypertext Markup Language
IBR	Introduction Based Routing
ICS	Industrial Control Systems
IDS	Intrusion Detection System
IEEE	Institute of Electrical and Electronics Engineers
IP	Internet Protocol
IPSec	Internet Protocol Security Encryption Scheme
IT	Information Technology
JIOR	Joint Information Operations Range
LSD-41	Landing Dock Ship, Class LSD-41
MA	Massachusetts
MB	Megabits
MCS	Machinery Control System

MITM	Man-in-the-Middle
MLC	Marine Logistics Group
MRC	Mission-oriented Resilient Cloud
NAV	Navigation System
NAVSEA	Naval Sea Systems Command
NFQ	Netfilter Queue
NGO	Non-Governmental Organization
NPD	Network Path Diversity
NSDI	Symposium on Networked Systems Design and Implementation
NSWC	Naval Surface Warfare Center
NY	New York
OH	Ohio
OMB	Office of Management & Budget
OODA	Observe, Orient, Decide, Act
OS	Operating System
PACOM	Pacific Command
PI	Principal Investigator
PLC	Programmable Logic Controller
QuanTM	Quantitative Trust Management
QTM	Quantitative Trust Management
REGRET	Reputation In Gregarious Societies
REST	Representational State Transfer
RISC	Reduced Instruction Set Computer
RM	Reputation Management
RSA	Rivest-Shamir-Adleman cryptosystem
S	Seconds
SAC	SOUND Administration Console
SAF/AQR	Secretary of the Air Force, Science, Technology, and Engineering Directorate
SAFE	Semantically Aware Foundation Environment for CRASH
SAR	Same As Report
SCS	SOUND Connection Service
SCS	Ship Control System
SDD	Software Design Document
SDS	SOUND Data Service
SecDef	Secretary of Defense
SHA-1	Secure Hash Algorithm 1
SIGOPS	Special Interest Group on Operating Systems
SIS	SOUND Identity Service
SMT	Satisfiability Modulo Theories
SOCOM	Special Operations Command
SOUND	Safety On Untrusted Network Devices
SPAWAR	Space and Naval Warfare Systems Command
SPS	SOUND Policy Service
SQL	Structured Query Language
SRS	SOUND Reputation Service

SSDD	System/Subsystem Design Description
SSF	SOUND Sensor Framework
STINFO	Scientific & Technical Information Office
STS	SOUND Trust Service
SUPPL	Simple Unified Policy Programming Language
TCP	Transmission Control Protocol
TM	Trust Management
TR	Technical Report
TTA	Technology Transfer Agreement
TUN	Tunneling Interface
TV	Trust Value
UAV	Unmanned Ariel Vehicle
USAF	United States Air Force
USENIX	The Advanced Computing Systems Association
US	United States
VA	Virginia
VDD	Version Description Document
VLAN	Virtual Local Area Network
VLC	a software multimedia player
VSE	Virtual Secure Enclave
WP	Wright-Patterson
XFRM	IP framework for transforming packets
XML	Extensible Markup Language
XMPP	Extensible Messaging and Presence Protocol